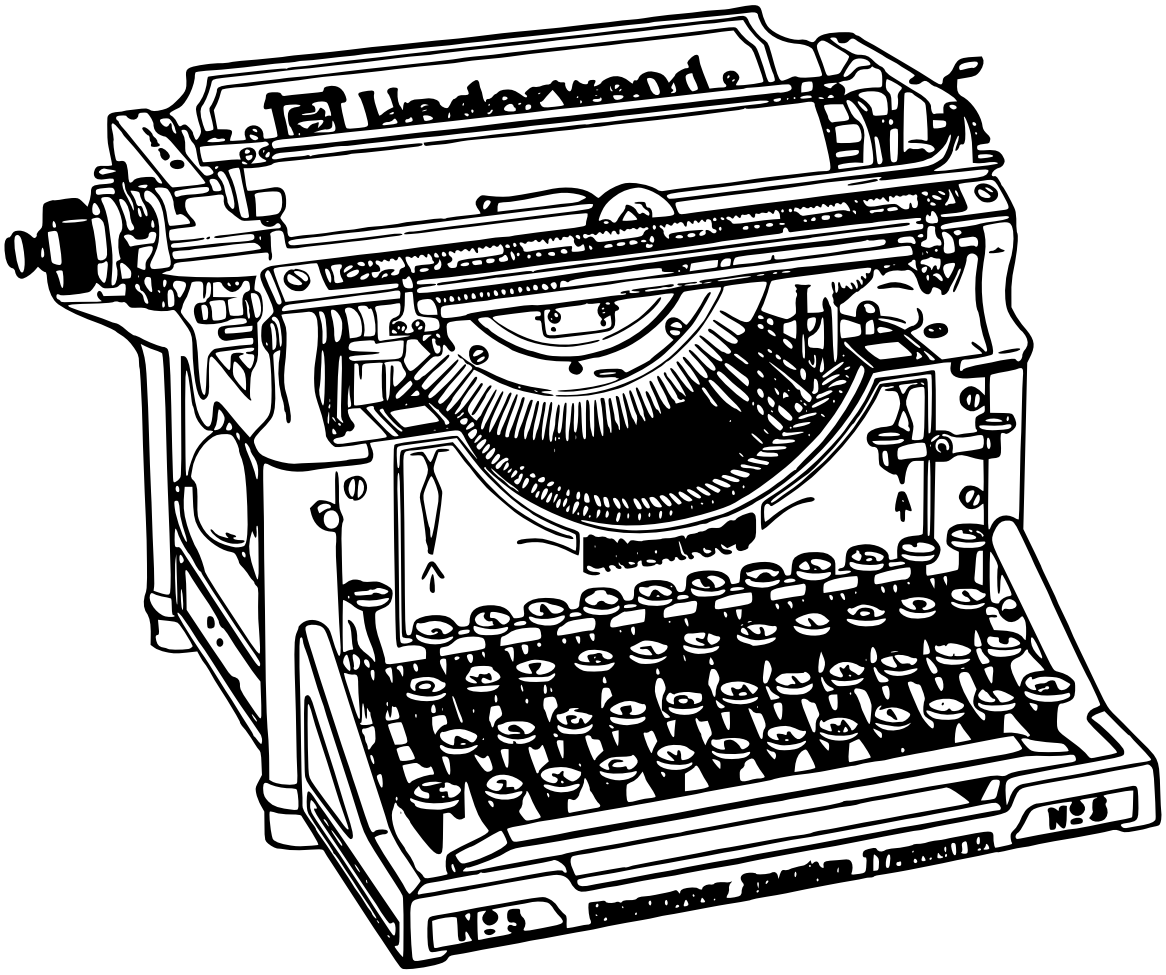
Pablo Rodríguez

Escritura con ordenador

Cómo la informática ayuda a la literatura



http://www.escritura-ordenador.tk

Escritura con ordenador

Cómo la informática ayuda a la literatura

Pablo Rodríguez

2016

http://www.escritura-ordenador.tk

A Pablo, porque la informática
acaba sirviendo.

Carlos, Edu, Nena, Óscar, Pablo
y Rosa: os podría llegar a ser
útil, aunque no lo parezca.

Computers are good at following instructions,
but not at reading your mind. [1]

Sumario

Sobre la licencia de uso

Este libro se ofrece con descarga gratuita en `http://www.escritura-ordenador.tk`. Esta obra no está bajo una licencia *Creative Commons* u otra licencia libre similar.

Mi intención es poner este libro a disposición de toda persona que quiera leerlo. Espero que sea útil. A cambio, sólo pido un sencillo pacto de nobleza. Creo que cualquiera estará de acuerdo.

- Si el libro te parece útil, recomiéndalo a quien le pueda interesar. Hazle llegar la dirección `http://www.escritura-ordenador.tk`. Así tendrá la última versión disponible. También sabré el número de descargas con bastante fiabilidad.

- Si consideras que tiene puntos mejorables, no dejes de decir cuáles son. Tampoco olvides de describir en qué deben mejorar.

- Si consideras que el libro es malo, agradezco la crítica. La única condición es que sea respetuosa con las personas.

Los comentarios son incidencias en `https://github.com/ousia/escritura-ordenador/issues/`. Tienes que estar inscrito en *GitHub*. Todos los comentarios son públicos.

Para que no haya incomprensiones, hago las aclaraciones pertinentes. Así nadie se lleva a engaño.

Únicamente se permite la descarga desde la página mencionada. Se permite el uso personal —no colectivo— de los archivos. Si quieres compartir el libro, envía http://www.escritura-ordenador.tk.

Dentro del uso personal, se incluye la impresión del archivo PDF en establecimiento público. En la página se explica cómo hacerlo. El establecimiento no está autorizado a vender el libro, sólo a imprimirlo a petición de la persona interesada. El precio de impresión y encuadernado no podrá ser superior al del mismo número de páginas y encuadernación de cualquier otro documento.

Una última aclaración para las entidades de gestión colectiva de derechos —Centro Español de Derechos Reprográficos, CEDRO—: no cabe el cobro de licencia por fotocopia de esta obra. Este título no se encuentra dentro de su catálogo.

No existe copia privada en fotocopiadoras de uso público por excepción reglamentaria (Real Decreto 1657/2012, artículo 3.4.a), https://www.boe.es/buscar/act.php?id=BOE-A-2012-14904#a3).

Por tanto, no pueden cobrar por esta obra. Ni por copia privada, ni por cualquier acuerdo o programa de licencias con su entidad —https://www.conlicencia.com—. Ustedes carecen de ella.

Prólogo

En *Ensayo sobre la escritura. Incomprensiones, falacias y mitos*[2], había pensado añadir uno o varios apéndices sobre la escritura con ordenador. Sin embargo, advertí que esos añadidos quizá deberían ir aparte. Su temática es realmente distinta, por mucho que se pueda presentar como complementaria. Además, los anexos podrían tener una extensión igual o superior a la del propio texto. Por ambos motivos, entendí que debían ser el contenido de un libro distinto.

Me gustaría hacer una advertencia sobre el estilo de estas páginas. Quienes lean pueden reprocharme que el tono es prolijo. Tendrían razón, es algo que persigo. Al explicar cuestiones prácticas de informática asumo que quien lee no tiene especiales conocimientos técnicos. La única manera de que lo pueda entender quien lea es explicarlo más y mejor. Lo siento por quienes sepan más, podrán leer estas páginas más rápido. O incluso puede que otro tipo de recursos les proporcionen una información más adecuada a sus capacidades e intereses.

28 de octubre de 2016

Introducción

Ya en la mitad de la segunda década del siglo XXI, reconozco que en muchos casos escribirás con ordenador. No te recomiendo que lo uses siempre para escribir. Porque perderás agilidad con las manos. Además de la caligrafía, que son aquellos rasgos de tu escritura que conforman también tu personalidad. Sin embargo, hay casos en que escribir a ordenador es casi una necesidad. Hacerlo a mano para luego pasarlo a ordenador sería un engorro, cuando no directamente un absurdo. Estas páginas pueden ayudarte en la tarea de usar un ordenador para escribir.

Este libro no es un libro de informática al uso. O no lo es en parte. Porque no sólo pretendo explicarte cómo se usa un programa. En el fondo, la explicación es el relato de un recorrido personal. No es tanto la narración de la búsqueda, como la descripción de sus resultados. De ahí el subtítulo *Cómo la informática ayuda a la literatura*. La pregunta que pretendo responder en estas páginas es cómo se puede escribir mejor usando un ordenador.

Existen herramientas profesionales con las que puedes lograr mejores resultados e invertir menos tiempo. Te pueden hacer más eficaz y eficiente en la parte en que lo necesites. No reduzco toda la escritura a ambas cualidades. En la parte más prosaica o menos creativa, te aprovecharán mucho ambas características. Las herramientas de las que te hablo no van a escribir por ti. Únicamente te ahorrarán algunos engorros de la mecánica de escribir. Esto es muy importante, aunque no

sea lo decisivo de la escritura. La finalidad de la informática es facilitar tareas que puedes automatizar. La escritura misma no es un automatismo, aunque muchas de sus tareas relacionadas sí pueden serlo.

Antes de pasar a explicar de qué se trata, creo que es importante que te sitúes en una nueva perspectiva. De otro modo, es fácil que lo que te cuente te suene a chino. La informática es realmente mucho más fácil de lo que parece. El problema muchas veces estriba en que quien explica no se hace cargo de las carencias de quien se esfuerza por entender. No por impaciencia o comodidad, la mayoria de las veces es por pura inadvertencia.

Te voy a contar una anécdota personal que recordaré toda la vida. Hace mucho tiempo —los dinosaurios se acababan casi de extinguir—, una persona me dedicó una hora para explicarme cómo funcionaba un *Macintosh*. Era un modelo de ordenador de *Apple*. Yo tenía experiencia con ordenadores con *MS-DOS* y creo recordar que *Windows 3.1* —no recuerdo exactamente el número de la versión—. Lo dicho, casi me encuentro con los dinosaurios. Sabía manejarme con ordenadores. Después de la hora, esa persona se fue y me dejó solo. Estaba en una sala común de ordenadores con bastantes más personas. No conseguía que funcionase nada. Pasé así un buen rato, creo que fueron cinco minutos de reloj. Hasta que me dí cuenta de que la persona se había olvidado de decirme que tenía que pinchar dos veces con el ratón. Ahora es lo habitual, pero con *MS-DOS* o *Windows 3.1* no era así.

Esa deficiencia en la explicación pasa por alto algo todavía más básico. Es muy difícil poder tener destreza de uso si no entendemos qué estamos haciendo en el ordenador. Por eso,

no debemos confundir qué hace el ordenador con lo que nos imaginamos que hacemos con él. Tenemos que entender qué estamos haciendo informáticamente. Aunque sólo sea para saber dónde nos estamos moviendo. O siquiera para conocer dónde estamos. Si no entendemos, el ordenador nos parecerá magia. Estaremos obedeciendo a ciegas dictados de no se sabe qué, ni tampoco por qué.

No se trata de que tengas conocimientos de arquitectura de sistemas. Has de entender qué haces. Es algo muy básico: que aprendas a usar un ordenador antes que a usar programitas. La informática son más procesos o métodos que opciones de un determinado programa. Si sabes usar un ordenador, podrás aprender a usar nuevos programas cuando te hagas con su lógica interna. Si sólo sabes usar programas, te descolocarás en cuanto los cambies. Incluso la desorientación podrá hacer mella en ti con una nueva versión del mismo programa.

No se trata de te vuelvas un apasionado de la informática. Como tal, la informática da para pocas pasiones: sólo son ceros y unos. La informática es interesante por lo que podemos lograr con ella. No tienes que estudiarla en profundidad. Sólo hay que tratar de comprender la lógica interna del ordenador. Porque así ganarás mucho. Entenderás mejor lo que haces. Te harás cargo de por qué el ordenador te da determinados fallos. En definitiva, sabrás dónde estás y dónde te mueves. Sin haber dedicado más tiempo a aprender, sino sólo habiéndote preocupado de entender mejor.

Parte de la motivación que me mueve a escribir estas páginas es el entusiasmo. O la admiración por lo que se puede llegar a lograr con ciertos programas. Te voy a poner un ejemplo rela-

cionado con el trabajo con textos. Después de haber dedicado tiempo a la tipografía digital, veo que la mayor parte de los libros editados tienen fallos de principiante. No es que tengan erratas, que por desgracia también las hay. Para las erratas, da lo mismo que hayas ganado el premio Nobel de literatura o que seas uno de los mejores autores del siglo —pasado, en este caso—. Las faltas son carencias de capacidades tipográficas profesionales que están al alcance de todo el mundo. Usando esos programas, los textos parecen de imprenta. Así no parecerán unas hojas con la tipografía predeterminada y las carencias habituales. Con la calidad profesional, el resultado es mucho más elegante y más legible. Aunque a veces esa calidad les falte también a los profesionales.

No te quepa la menor duda de es mejor aprender a trabajar informáticamente con textos. Con nivel del usuario que sabe aprovechar las posibilidades que brinda la informática. Así no dependerás de otras personas. El trabajo te resultará mucho más fácil cuando hayas adquirido destreza con la informática. Cuando sabes tú, puedes corregir errores de otros. También podrás evitar que los demás te cuelen errores. La calidad de tus documentos será mucho mayor. Porque al menos su presentación será mucho mejor. Sencillamente es aprender a trabajar mejor. Toda la tarea que trato de describirte es usar mejor la informática.

En el fondo, la idea que quiero transmitirte es que podemos aprovecharnos de capacidades extremadamente útiles de programas que solemos reservar a los especialistas en informática. Una muy buena amiga mía busca en internet la solución de problemas informáticos con una convicción básica: seguro que

no es a la primera persona que le pasa. Tampoco somos los primeros que usamos ordenadores en el planeta. Por eso, es fácil que haya herramientas que hagan lo que intentamos hacer, mucho mejor de lo que pensamos. Somos enanos en informática, pero podemos subirnos a los gigantes para otear la realidad desde sus hombros.

OTRA PERSPECTIVA

1 Independencia informática

Como sabes, uso *Linux*. Desde hace casi tres lustros es el único sistema operativo que tengo en mis portátiles. Creo que es un sistema fiable y bueno. Pienso que es muy útil. Por supuesto, en el trabajo uso lo que hay. De momento, es *Windows*. Como a lo mejor no puede ser de otro modo, cada vez noto más las carencias de *Windows*. De las últimas experiencias que he tenido, destacaría las siguientes:

- Ayudando a un amigo, tuve que instalar *Windows* en su ordenador. Me llevó bastante menos instalar *Linux* completo en mi portátil, que *Windows* en el suyo. Con *Windows* todavía tenía que instalar todos los programas. Mi portátil era unos cuatro años más antiguo que el suyo, por lo que no tenía tanta capacidad de proceso y era más lento.

- Mi ordenador portátil tiene diez años[3] y funciona perfectamente con una versión reciente de *Linux*[4]. Me temo que para instalar *Windows 10* tendría que cambiar de ordenador primero[5].

- El ultraportátil desde el que escribo este libro tiene un lustro. Gracias a cambiar *Windows Vista* por una versión reciente de *Linux*[6], puedo usarlo para escribir. No es un ordenador potente, pero sirve perfectamente para hacer lo que describo en este libro.

Sin embargo no pretendo hablar aquí de sistemas operativos. Creo que cada cual debe usar lo que más le convenza. Por eso, entiendo que las soluciones informáticas deben ser neutras. La neutralidad debe ser la independencia de sistemas operativos. Así trabajas con estándares, no con productos comerciales de un signo u otro.

Esa independencia de un determinado programa o una determinada plataforma es lo que debería caracterizar el uso de ordenadores. Piensa en internet. Internet funciona en gran parte por un pequeño detalle. El código en el que se escriben los documentos está normalizado. Se llama *hyper text markup language*. Es la terminación `.html` que tiene un determinado tipo de archivos. Por eso puedes ver internet con el navegador que te funcione mejor, no con el que desarrolle *Microsoft*, *Apple*, *Google* o *Mozilla*. La normalización es muy importante.

La estandarización también se puede mostrar en negativo. Te darás cuenta que cada vez hay menos aparatos que funcionen con pilas normalizadas, aunque sean recargables. Tienen baterías propias. La mayoría de las veces te quedas sin un aparato —por ejemplo, un reproductor de música, un lector de libros electrónicos o un teléfono inalámbrico— porque se agota capacidad de carga de la batería. Si tuviese una pila normalizada, la cambiarías y ya está. Es cierto que en algún caso puedes buscarla en internet y ponérsela tú. Pero no es lo que está pensado. De hecho, si le cambias la batería —puede tener una garantía más corta que el resto del equipo[7]— antes de que termine el plazo de garantía, te puedes encontrar con que la garantía de todo el equipo esté anulada por ese cambio.

De *Linux* no me interesa el sistema operativo en este momento, sino su modelo de desarrollo. En realidad, *Linux* es el núcleo de un sistema operativo. Sólo con el núcleo apenas podrías usar el sistema. Las versiones completas —distribuciones es el término preciso— tienen además del núcleo, muchas más cosas, como al menos un entorno gráfico. Como tal, *Linux* es una marca registrada en casi todo el mundo por Linus Torvalds. No se trata de esa marca contra otra marca registrada. Es un estilo de desarrollo informático frente a otros sistemas de desarrollo informático más precarios.

Como te he dicho antes, lo relevante de trabajar con un ordenador es la automatización de tareas en procesos informáticos. No se trata tanto de usar el programa determinado. Por eso es importante la existencia de estándares. Te pongo un ejemplo. La mejor manera de compartir documentos que no van a ser modificados es el PDF. Porque la página que ve quien imprime es exactamente la misma que ve quien la ha creado. Las tipografías serán las mismas[8]. Las páginas o las líneas no se partirán de manera distinta. Imagínate la gran ventaja de poder estar seguro de lo que imprimes. Cuántos libros o documentos cambian por sólo pasarlos de un ordenador a otro. No es que el texto sea distinto, es que cambia su presentación. Por cierto, lo de que los archivos PDF no son modificables, es sólo un poco de desconocimiento. Son modificables, aunque no sea tan fácil de modificarlos como los documentos de un procesador de textos.

Además, PDF es un estándar ISO —el 32000–1, para ser más exactos[9]—. Gracias a que es un formato público, cualquiera puede escribir programas que lean archivos PDF. También cualquiera puede escribir programas que generen archivos PDF.

Por poner un ejemplo con lectores de PDF, *SumatraPDF* es un lector de documentos PDF [10] mucho más eficiente —necesita menos recursos y va más rápido— que la última versión de *Adobe Reader* [11]. Gracias a que la especificación es pública, no tienes que comprar el programa de *Adobe* para generar documentos PDF.

Linux es una metáfora de lo que realmente es un modelo informático colaborativo. El código de *Linux* puede venderse tal como está. Aunque es mucho más rentable vender servicios. Habitualmente son empresas los que buscan esos servicios para poder trabajar con *Linux*. La cooperación está en aportar código con mejoras y parches para fallos que pueda haber. Incluso un informe de un fallo perfectamente delimitado puede ser extremadamente útil. Porque los autores del programa pueden saber con precisión qué es lo que falla y arreglarlo.

Este sistema de cooperación puede entenderse de dos modos. Un modo viene de la teoría de juegos. El modelo de desarrollo cooperativo es un juego de suma positiva. No tiene que perder alguien para que pueda ganar su contrario. Un juego de suma positiva es en el que todos ganan, porque no hay confrontación. Habitualmente, una venta es un juego de suma cero, porque quien compra ha de empobrecerse para enriquecer al que vende. En este caso no es así. ¿Cuál es el incentivo? La ganancia económica inmediata está claro que no es el único incentivo. Hay otra serie de incentivos, incluso económicos, que no son la monetarización inmediata. Para que no parezca que quede en el aire, te planteo una pregunta: ¿crees que dedicarse a escribir sólo cabe hacerlo por dinero?

Los sectores de la economía también permiten explicar cuál es el sentido de un modelo de desarrollo cooperativo. Los sectores son tres: primario de materias primas, secundario de productos manufacturados y terciario de servicios. Si bien son necesarios los tres sectores, una economía es más rica cuanto más desarrollado tiene su tercer sector. Cuántos más servicios ofrezca una sociedad, más riqueza genera. Tomemos la informática. Los programas informáticos puedes verlos como la materia prima en una sociedad digital. El trabajo logrado con esos programas lo puedes considerar su producto. Los servicios serían los que puedes ofrecer a otros con los programas para realizar trabajos.

Cuantas más barreras haya para la obtención de materias primas, más lentamente se desarrollará esa sociedad digital. Con un ejemplo básico, un programa de generación de libros electrónicos. Tienes el programa —`pandoc`—, que te permite generar libros electrónicos. Puedes ofrecer tus servicios a escritores. O como lo que pretende este libro: dar claves para que otros puedan crear. ¿Con qué crees que se genera más riqueza: vendiendo el programa o consiguiendo que mucha más gente publique sus escritos?

2 ¿Son programas gratis?

Linux es un ejemplo de lo que se denomina en inglés *free software*. «*Software* libre» creo que es una mala traducción. Porque acaba apuntando a algo así como a programas gratis, que es lo que entiende la mayoría de la gente. En inglés puede entenderse así, si bien se suele añadir una aclaración: *free as in free speech, not in free beer*. Lo que quiere decir: «libre como en libertad de expresión, no como en libertad de barra».

Creo que «programación informática libre» refleja mejor la realidad a la que se refiere el modelo de desarrollo cooperativo. Pero antes de entrar en ese punto, quiero aclararte algo importante. Muchas veces el código de la programación libre es gratuito, pero quedarse sólo con eso es empobrecer totalmente la realidad a la que se refiere. La programación libre no son sólo programas gratis. Ni siquiera son principalmente programas gratis.

Antes de deshacer la incomprensión, quiero demorarme brevemente sobre la gratuidad de los programas. Reconozco que en una sociedad que tiene una situación económica al borde del colapso, que existan programas gratuitos es muy importante. Por varios motivos. Por el ahorro económico que supone. Por la alternativa a la copia no autorizada de programas, con la que sólo se contribuye a la extensión de su monopolio[12]. Por la mayor flexibilidad de soluciones informáticas. Por la independencia respecto a un único vendedor.

La gratuidad no es lo más relevante. Porque si lo tomas como lo más relevante, al final no acabas de ver otra cosa. Ése es el problema de muchas empresas. Poniendo un ejemplo: *LibreOffice* es un conjunto de programas ofimáticos del mismo rango que *Microsoft Office*. Que sea gratis puede ser importante, pero no podemos olvidar otros factores. No funcionan exactamente igual y a la gente le lleva tiempo aprender lo novedoso. La conversión de documentos puede ser errónea en algún punto. Tiene fallos de programación y funcionalidades no desarrolladas. Es distinto. Que no te cueste nada, no te debe cegar para tirarte a la piscina. Porque puede estar vacía.

Contra lo que puedas pensar, no tengo ningún problema con *LibreOffice*. Apenas lo he usado, porque tengo otro modo de trabajar —creo que más claro—. Cuando los veo, procuro informar a los desarrolladores de los fallos. Quiero mostrarte el error de la decisión anterior. La que lleva a tirarte a la piscina al grito de gratis, cuando no has visto siquiera si hay agua. Lo voy a hacer con un ejemplo. Supongamos que estás con más personas trabajando en una sala. De repente, alguien se acerca y pone en el centro un cuenco con caramelos. Son gratis. Esa persona pone más caramelos en el cuenco cuando se van a acabar. De lo que no nos podemos quejar es que con el cuenco no haya una bandeja con pastelitos. Si en un sistema cooperativo no aportas nada, seguirá habiendo lo que te encontraste al principio.

Ahora te lo cuento con la programación libre. Si pensamos que el código se escribe sólo, si esperamos a que nos lo den todo, acabarán dándonoslas todas. Un sistema cooperativo exige colaboración. Si ves un fallo, al menos debes informar de él. Si aportas un parche que lo solucione, mejor porque será más rápi-

do. Lo mismo sucede con las mejoras. Pero si vives como si las cosas se hiciesen solas, no has entendido nada. Además, cuanta más gente siga ese ejemplo, más fácil es que el programa deje de desarrollarse. Lo malo es que muchas empresas piensan que son programas gratis. Por no hacer, ni se molestan en actualizar a las nuevas versiones. Vaciarán el cuenco de caramelos, pero no traerán ni una botella con agua del grifo. Su modelo no es de escasez, sino de aceleración de la escasez, de desertización.

Todo esto te lo cuento para que veas una nueva perspectiva en la que situarte. Desde luego, espero que tengas claro que no es la de todo gratis. Es quizá que la unión hace la fuerza. Referido a la programación informática libre —también llamada de código abierto, *open source*—, creo que hay dos frases que te pueden ayudar a entender a qué me refiero. Son dos frases independientes, no expresan grados de desarrollo.

I came for the price. I stayed for the quality.

I came for the quality. I stayed for the freedom.

Esto es otro nivel al de programas de pago o gratuitos. No porque exista un mantra espirituoso y difuso. Sencillamente con cooperación se consigue lo que de otro modo no se conseguiría. Quizá por falta de incentivo económico. Piensa en una parte de internet, la gran malla mundial. Esto es, *the world wide web*. Gracias a que Tim Berners-Lee no quiso vender, sino distribuir sin restricciones, el protocolo de transmisión de hipertexto —como comienzan las direcciones de internet, `http`[13]—, internet puede ser hoy un grandísimo escaparate.

Jugar en esta división implica colaborar. En la medida de nuestras posibilidades. El programa que trato en estas páginas —pandoc— es un ejemplo de la mejor calidad de la programación libre. Por supuesto, la calidad la tienes que juzgar por sí misma. Si algo falla o falta, no es menos importante si es programación libre. Sin embargo, parte de esa exigencia de mejora revierte en ti. En cada uno de nosotros. Informa de la falta o fallo. En inglés. Si no, será necesario que alguien traduzca tu informe. Aporta un parche con la corrección del error o la nueva funcionalidad. Si el proyecto acepta donaciones, dona. Las tres cosas, siempre en la medida de tus posibilidades. Por supuesto, todo es libre. Si no quieres, no lo hagas. Aunque entonces, ni te quejes, ni te sorprendas de que nada cambie en el código.

3 Un nuevo horizonte

En este capítulo quiero explicarte algo sencillo. No es tanto un conocimiento técnico, como un cambio de perspectiva en el modo en que trabajas con ordenadores. Es algo realmente simple. Al principio te costará verlo, porque la informática te hace imaginar la interacción con un ordenador de una determinada manera. La imagen visual a la que ahora me voy a referir, es sólo una metáfora. Todo lo que pretendo contarte es que hay otros modos de trabajar con el ordenador distintos de esa metáfora. Son modos de trabajo más eficientes, porque usarás el ordenador de un modo más directo.

La informática son ceros y unos, no tanto formas y colores. Por eso, la imagen visual es una metáfora, una ficción. No es un fraude o un engaño. Simplemente es una simulación, un añadido a un sistema operativo para que quien use el ordenador sienta mayor comodidad en su manejo. La metáfora consiste en lo que se llama escritorio o sistema de ventanas[14]. Esa metáfora consiste en simular que el ordenador es como una gran mesa en la que tienes diferentes objetos con los que puedes interactuar. A la mayoría de gente le cuesta entender que el escritorio no tiene que ser el único modo de trabajar con un ordenador. Porque la imagen visual es muy fuerte. Aunque, para que quede claro en lo que resta de libro, puntualizo un detalle que creo que es importante. Cuando enciendo mi ordenador, tengo un sistema de ventanas, un escritorio. Para muchas cosas es muy cómodo.

Sin embargo, uso la línea de órdenes, que tiene unas capacidades que son imposibles de alcanzar visualmente.

Aunque creas lo contrario, la línea de órdenes es más fácil de manejar y de entender que el ratón con las ventanas. Exige que aprendas antes, pero una vez que has aprendido es más rápido. Aprender no es tan difícil como puede parecer al principio. Así puedes hacer cosas nuevas, imposibles o mucho más difíciles de lograr en el escritorio.

Te pongo un ejemplo. Imagínate que tienes que explicar a alguien por teléfono cómo puede copiar las fotografías de imágenes de las vacaciones de una carpeta en el escritorio a una memoria externa. Para facilitar las cosas, en esa carpeta hay también vídeos y música, que no quiere copiar. Si la otra persona lo tiene que hacer visualmente, creo que es muy díficil explicarlo bien. Con la ventana o línea de órdenes sería tal que así[15]:

```
cd Desktop/multimedia &&
cp -i *.jpg /mnt/TOSHIBA/vacaciones
```

Si no entiendes de lo anterior, te sonará a chino. Pero sólo mientras no lo entiendas. Si te lo explican bien, verás que es muchísimo más sencillo. Porque la única manera de explicarlo en el modo visual es hacer un vídeo. O estar delante de la persona y hacerlo en pantalla. Por teléfono, tendrías que relatar una pequeña odisea.

Los ejemplos podrían ser infinitos. Parte del problema está en el respeto a lo que *Windows* llama símbolo del sistema, que es la ventana de órdenes. Una pantalla negra con letras grises impresiona. Por si te sirve de consuelo, la mía es de fondo blanco

con el texto en negro. Lo cambio porque así me resulta más legible. ¿Da miedo porque no ves lo que haces? Es cierto que no ves lo que haces. Pero tampoco con las ventanas ves realmente lo que haces en muchos casos.

Te voy a contar un ejemplo que muestra que no siempre ves lo que pasa con las ventanas. Mucho menos, lo puedes entender. Es una anécdota propia. No sé por qué extraña razón, mis reproductores de música ordenan los archivos según orden de llegada. Siempre que copias una carpeta o varios archivos, no se copian necesariamente por orden. Eso normalmente no se nota, porque se ordenan luego correctamente. Lo que pasa es que si no hay otro criterio de ordenación que el instante de grabación, ahí si que se nota. Los archivos más grandes tardan más en copiarse. De una carpeta con un disco, ves que el orden del disco ha cambiado.

Tardé bastante en darme cuenta de qué fallaba. Hasta que advertí que tenía que copiar las canciones por orden *una a una*. O podía hacer que el ordenador lo hiciese por mí. No es cuestión del sistema operativo —de *Linux*—, sólo lo puedes hacer con una orden escrita. Porque necesitas un bucle, esto es, una orden repetitiva. La alternativa es clara: o lo repites tú, o lo repite el ordenador. La informática es buena para automatizar tareas. La repetición es algo muy fácil de automatizar. En este caso, hay que decirle al sistema lo mismo que harías tú: que copie el primer archivo, luego el siguiente, así hasta que haya copiado todos[16].

Lo que intento describir no es una colección de anécdotas personales. Ni de casos más o menos concretos. Son sólo ejemplos con los que trato de explicarte que la línea de órdenes es

una posibilidad importante en un ordenador. No está reservada a quienes programan. Es para todo el mundo. Es el modo de uso lógico. Es un enfoque lógico a diferencia del uso visual, de la metáfora del escritorio.

La diferencia de modos o de usos —lógicos y visuales— se puede comparar con las diferencias de comunicación. No es un análisis perfecto, es una imagen que creo que puede ser útil. El modo lógico es como hablar un idioma, mientras que el modo visual es como tratar de comunicarse por señas. Comunicarse por señas es acercarse la mano a la boca para expresar «comida», no es la lengua de signos. El modo lógico usa palabras y construcciones con esas palabras. Tiene que observar una ortografía —un modo correcto de escribir esas palabras— y una sintaxis —no puedes construir frases de cualquier manera—. No se me olvida que un código no es una lengua. Es sólo el ejemplo.

El modo visual es deíctico por definición. Tiene que señalar objetos, sean archivos, carpetas, diálogos, menús... No por casualidad a la imagen del ratón en pantalla se le llama puntero —*pointer* en inglés—. Como en realidad, ir señalando cosas resulta una comunicación muy pobre, muchas veces habrás de señalar palabras. Entre otros, en eso consiste la selección de opciones de un menú. La carencia básica del modo visual es que hace muy difícil la abstracción, la generalización. Porque lo visual y lo que se indica es necesariamente lo concreto. Sin embargo, la utilidad de la informática está —como veremos con más claridad en el próximo capítulo— en poder abstraer.

Por supuesto, hablar un idioma exige mayor aprendizaje. El enfoque lógico es más exigente. Tendrás que aprender más. O tendrás que aprender en serio. Aunque quiero aclararte algo

sobre lo que ahora se llamarán «competencias informáticas» [17]. Pinchar con un ratón las veces que sea necesario no es propiamente saber de informática. Es esperar a ver qué pasa. Es más intuitivo, suele decirse. Quizá sólo exige menos conocimientos. Porque no sabes qué pasa al pinchar. No entiendes cómo funciona ni el programa, ni el ordenador. Simplificar el funcionamiento no siempre supone simplificar la informática. A veces sólo se simplifica tu mente. En el fondo, el enfoque lógico expresa la necesidad de entender si quieres enterarte de qué está pasando. Porque te permite comunicar con el ordenador de tú a tú.

Como con toda lengua desconocida, al principio te parecerá imposible aprender las órdenes escritas. Con explicaciones sencillas, irás viendo el modo en que puedes entender la estructura básica de esa lengua, de esas órdenes. La línea de órdenes es mucho más sencilla que una lengua. Porque no necesitas tanto como para el nivel básico de una lengua. La estructura es mucho más simple. Tienes que saber sólo las palabras de aquello que uses. Normalmente será muy poco. Cuando necesitas algo nuevo, buscas sus palabras y ya está.

En definitiva, este capítulo sólo quiere transmitirte una idea básica. La ventana de órdenes es tu gran aliada. No es un campo minado. Tienes que ganar confianza con ella. Sólo tienes una manera de lograrlo: usarla todo lo que puedas. Tendrás que aprender algo, pero en realidad es muy poco. Luego practicar con el uso. Sólo así podrás sacar más partido al ordenador del que quizá habías pensado nunca.

LA EDICIÓN DIGITAL

4 Más allá del procesador de textos

Eres demasiado joven —con esa afirmación intuyo que soy demasiado viejo—, pero supongo que habrás visto alguna vez en tu vida una máquina de escribir. Incluso manual, de las de siempre. Como curiosidad, hace un par de años ví una noticia de que había escritores que volvían a las máquinas de escribir manuales. Entrevistaban brevemente a un señor que las reparaba en California —creo recordar, en todo caso era Estados Unidos—. Decía que le contaban que la máquina de escribir no tiene distracciones. Pero no deja de ser una curiosidad. Las distracciones de un ordenador sólo podrás superarlas con disciplina a la hora de escribir. A la hora de trabajar con cualquier ordenador —incluidos los mal llamados teléfonos inteligentes— necesitarás siempre ser dueño de tu atención.

La evocación de la máquina de escribir no es casual. Precisamente, es la imagen visual con la que funciona el procesador de textos. Cualquier procesador de textos supone que en pantalla tienes una hoja en blanco que vas llenando a medida que tecleas. A diferencia de las máquinas de escribir clásicas, el ordenador tiene memoria. Por tanto, el procesador de textos tiene memoria. Al igual que la máquina de escribir, lo que se dibuja en la pantalla es lo que tienes. No es que siempre funcione. Es que no tienes otra manera de saber qué tienes si no es con la imagen de la pantalla.

El procesador de textos tiene varios tipos de letra y de tipografías. Justifica los textos también en el margen derecho —algo

imposible para una máquina de escribir—. Incluso permite partir palabras al final de la línea, para que el espacio entre palabras sea más homogéneo. Puede hacerlo de modo automático, aunque la mayoría de la gente no sabe cómo usar esa capacidad. Esa simulación tiene sus límites. No tanto en lo que muestra, como en lo que puedas hacer con el texto informatizado.

Te voy a mostrar ejemplos de personas a las que les he ayudado con sus documentos de procesadores de textos. Son problemas más o menos graves, derivados de haber trabajado con un enfoque visual. Aprender a trabajar así cuesta tiempo —a veces incluso dinero—. Pero si nadie te ha enseñado a hacerlo mejor, tienes que sufrir las limitaciones de ese método de trabajo. Quizá no habría costado más aprender de otro modo. Sin embargo, nos preocupamos poco de aprender cómo funciona un ordenador. No es tan difícil. A la larga, creo que el aprovechamiento es mucho mayor.

Quizá la limitación estrella puede formularse con una pregunta. Antes de plantearla, piensa que no me refiero a artículos, sino a libros completos. De más de doscientas páginas impresas. La interrogación es muy sencilla: ¿cómo cambiarías la tipografía de un documento a *Palatino*? La grandísima mayoría de la gente no dudaría en seleccionar todo el texto y aplicar la tipografía *Palatino*[18]. Así es fácil que haya cosas que se queden fuera, como las notas al pie. También puede haber pasajes con otra tipografía, que no deban cambiarse y que se modificarán. Si tenemos muy mala suerte, podemos llegar a perder negritas y cursivas.

No sé si consigo mostrar en qué consiste la chapuza de este modo de hacer las cosas. No es tanto que no sea el modo propio

de hacerlo, sino que habrá fallos seguro. El modo propio de hacerlo con un procesador de textos sería usar estilos. Habría que definir qué tipografía va a usar en los estilos del documento. Aunque te parezca lo mismo, en absoluto lo es. No son iguales los resultados. Pero mucho antes, hay algo que falla. Es no advertir la diferencia entre un enfoque visual y uno lógico. La informática son órdenes. El enfoque lógico afirma en este caso: la tipografía del documento es *Palatino*. El enfoque visual ordena realmente: aplica *Palatino* a lo que está seleccionado.

Las órdenes del ejemplo no están al mismo nivel. Por eso, si tuvieses que volver a cambiar la tipografía del texto, es posible que te encontrases nuevos errores. Sobre todo, lo más grave es que piensas que estás haciendo una cosa, cuando realmente estás haciendo algo similar. En el fondo, harías lo mismo que con la cursiva de un título de una obra o una palabra en un idioma extranjero. Defines un pasaje y le aplicas una propiedad tipográfica. En vez de la cursiva, es *Palatino*. No es una palabra o una expresión, sino todo el texto. Espero que lógicamente quede claro que está mal y por qué. Por eso cabe esperar que haya errores, porque es el camino equivocado.

Otro ejemplo es añadir encabezados y pies de página a un documento en un procesador de textos. Lo he hecho únicamente con *Microsoft Word*, varias veces. Lo que buscaba era poner el título del capítulo en los encabezados impares y el título de la sección en los pares. Es una orden sencilla. No sé por qué extraño motivo, siempre que lo he hecho, tengo que repetirlo. Una vez para fijarlo y otra para que los encabezados no estén sólo según un capítulo. Comentándolo al tiempo que lo hacía con una amiga, me decía ella: «*Word* carece de una lógica clara».

No sé si la lógica me falla a mí o al programa —a lo mejor ya está solucionado—. Pero si no ves lo que haces, si no puedes ordenar directamente al programa, es más fácil que no puedas sabe qué falla, ni tampoco por qué.

El penúltimo ejemplo es algo que es más díficil de hacer con un procesador de textos. No es imposible, pero exige un enfoque totalmente lógico y es más engorroso hacerlo en un procesador de textos. Bien, supongamos que tienes un documento. Puede ser un artículo o una tesis doctoral completa. Imagínate que citas unos cuantos libros. Además incluyes una cantidad importante de expresiones en lenguas extranjeras. Por si no has adividado, se supone que ambas deben ir en cursiva. Ahora tienes que cambiar unas —los títulos o las expresiones extranjeras— de cursiva a negrita o a comillas. No puedes cambiar todo. Porque una mitad estaría mal cambiada. Con un procesador de textos, es casi imposible no tener que ir buscando las cursivas y cambiarlas *una a una*.

La mayor limitación del procesador de textos es la falta de abstracción. La imagen de la hoja en blanco es la definición de su tarea. Si un procesador de textos tiene esos límites, es fácil que no funcione bien cuando intentas pasar a otros contextos. No es un problema de mala programación, probablemente. Es la simple imposibilidad de afirmar la contradicción: algo no puede ser y no ser al mismo tiempo y en un mismo sentido[19]. El procesador de texos podrá generar documentos similares. Serán documentos basados en el modelo de la página, como PDF. Sin embargo, fallará cuando haya que prescindir del modelo de la página. El ejemplo más claro es el formato de las páginas

de internet, o el normalizado para libros electrónicos —con extensiones `.html` y `.epub`—.

La dificultad de convertir a HTML te puede sonar muy rara. Puedes preguntarte legítimamente si acaso el texto mismo no está en el documento del procesador de textos. Por supuesto que está, pero está con muchísima más información irrelevante, que no aporta nada a la hora de que un navegador lea el contenido de la página. Cuanta más información haya, más recursos de proceso de datos gastas. Si la información es irrelevante, estás tirando esos recursos. Además, habrá cosas que necesariamente se convertirán mal.

En definitiva, un procesador de textos es un modo muy precario de trabajar un texto a ordenador. Porque el texto no es el que ves en pantalla, sino el que contiene el documento. Ése no te lo muestra el programa. En esa cortina impuesta por el procesador de textos, sólo consigues que haya una distancia y dificultad añadidas que no mejoran ni facilitan el trabajo a la larga.

5 *El enfoque lógico*

El enfoque lógico es la manera mejor de que trabajes con textos. Ahí hablas al ordenador de adulto a adulto. ¿Por qué? Simplemente porque el programa no te oculta lo que tienes. Lo que te muestra no es el resultado final. En pantalla está lo que realmente tienes en el documento. Si me permites la expresión, es el texto en sí. Por favor, no lo confundas con el en sí del texto, no tiene nada que ver —aunque sólo sea por respeto a Platón—. El texto en sí son tus palabras con el formato que sea necesario. Pero el formato necesario es el formato necesario del texto, no del programa. Un modo de trabajo lógico con el texto no añade códigos o etiquetas superfluas.

Te muestro un ejemplo, para que vayas viendo cómo es en la práctica el enfoque lógico. Primero te muestro el resultado de la frase:

> Esto es *énfasis* y esto es `código`.

El código que he tecleado en mi documento para tener esa frase es el siguiente:

```
Esto es _énfasis_ y esto es `código`.
```

No te dejes engañar por la apariencia extraña, por el mero desconocimiento. En realidad, no es nada. Es mucho más claro y tecleas problablemente menos. Por supuesto, necesitas acostumbrarte a escribir así. Cuando te hagas con eso, verás cómo es mucho más cómodo. El tiempo necesario de aprendizaje es

mínimo. Con un ejemplo tonto entenderás que no es tan difícil. Si yo lo entiendo, desde luego que es seguro que cualquiera puede entenderlo. El ejemplo es el estudio del griego en la enseñanza media. Para fijar el alfabeto griego —lo primero que se aprende—, el primer ejercicio es escribir palabras en tu idioma con letras griegas. Así se fijan las letras. Es algo que dominas en unas horas. O como mucho, en unos días. Con el código es igual.

La ventaja de tener enfrente el texto puro con el formato que sea necesario es la visibilidad. Tú sabes perfectamente qué hay, porque realmente no hay más. Si algo no funciona, lo tienes a la vista, delante de los ojos. Piensa en los dos ejemplos del comienzo del capítulo. El primero es puramente visual. Así trabajas con un procesador de textos. Me dirás que es más cómodo. A lo mejor no tanto. En la cursiva, no sabes seguro si el espacio de después de «énfasis» tiene cursiva o no. No es una ninguna tontería. Si ese espacio posterior está en cursiva y empiezas a escribir ahí, escribirás en cursiva. Me he encontrado con ese error no pocas veces. La gente a veces no sabía qué fallaba. Además, un espacio en cursiva puede ser más pequeño que uno normal. Dependiendo del texto, puede tener un resultado nefasto. Lo mismo pasa con la palabra en máquina de escribir. Tienes que tener muy buena vista para saber si el punto posterior está en máquina de escribir o no. O más que buena vista, tienes que tener bastante capacidad de observación.

Aunque te parezca increíble, el primer ejemplo del texto con formato final es una parte de un documento perfectamente opaco. Vas ciego, porque no ves qué tienes. Otro ejemplo muy común son los saltos de línea y los espacios. Excepto en un

procesador de textos, los espacios y las líneas en blanco consecutivos cuentan como uno. Quizá si no estás acostumbrado a eso, puedes pensar que es una mala idea. Entonces, no adviertirías la gran cantidad de errores que evita. Para detectar un espacio de más en un texto hay que tener un ojo muy entrenado. Incluso así, se te pasarán muchos. Algo parecido pasa con las líneas en blanco. Evita muchos más errores que contar cada línea en blanco como un nuevo párrafo vacío.

Como estás acostumbrado al enfoque visual —a un procesador de textos— y eres inteligente, te vendrá a la cabeza una pregunta. No es otra que cómo puedes introducir espacios o saltos de párrafo más grandes que lo habitual. Antes de seguir con la pregunta, debes considerar algo. Los espacios entre palabras y entre párrafos deben ser homogéneos. Si crees que tienes que usar espacios o saltos de párrafo más grandes, estás haciendo una chapuza.

Te voy a poner dos ejemplos con los que entenderas por qué es una chapuza. El primer ejemplo es de espacios. El sangrado de la primera línea de un párrafo —o del párrafo entero— no se debe hacer con espacios. Debes usar el sangrado de primera línea o el margen izquierdo del párrafo. Incluso con un procesador de textos deberías hacerlo de este modo. Así, cuando inicias un nuevo párrafo, la primera línea o el párrafo concreto está bien sangrado. El segundo ejemplo es con los saltos de párrafo. Antes o después de un título sólo debe haber un salto de párrafo. El título tendrá más espacio con las propiedades de espacio antes o después del título que se trate.

En el fondo, con el enfoque visual lo único que se tiene es un documento ciego. Es cierto que tienes el resultado, pero no

sabes qué es lo que hay realmente en el documento de texto. Es precisamente porque texto y formato están mezclados de tal manera que son indistinguibles. Todo es una amalgama, que es fácil que te confunda y te provoque errores. Porque, aunque no quieras, tienes que trabajar al mismo tiempo con texto y formato. No puedes separarlos y centrarte sólo en el texto. Porque en el mejor de los casos, tendrás demasiadas distracciones. El enfoque visual dificulta el trabajo puramente lógico con textos.

6 El asistente tipográfico

Quizá la última frase del capítulo anterior te ha dejado inquieto. Porque no está claro qué es un trabajo puramente lógico con textos. Me refiero a que te puedas centrar en escribir, sólo pendiente de las palabras que componen tu texto. Eso supone poder decir qué es cada cosa dentro de éste. Por ejemplo, títulos, notas al pie, énfasis, listas, citas independientes... Sin necesidad de decir cómo quedan al final, sólo sabiendo qué son.

Para que entiendas qué es ese modo de trabajar, te pediría un pequeño viaje en el tiempo. Un siglo hacia atrás. No exacto, porque Europa estaba sufriendo la que entonces conocía como la Gran Guerra. Pongámonos en 1910. Supón que eres alguien que se dedica a escribir y a publicar. Quizá algunos autores escribirían entonces a mano. Aunque para facilitar el trabajo de edición del libro, ya se usaban máquinas de escribir.

Con una comparación puedes ver la ventaja de poder trabajar de otro modo. Seguro que hace un siglo se publicaba mucho menos. Sin embargo, es más que probable que lo que se publicaba estuviese mucho mejor editado. Dicho llanamente, era más legible. El motivo no es ningún secreto. Se trataba de la división del trabajo. Quien escribía no editaba el texto. Editar no es en este caso publicar, sino componer tipográficamente el texto —darle formato—. El tipógrafo era quien se manchaba las manos de tinta componiendo cajas con el texto ya escrito. El texto ya redactado lo recibía en un manuscrito mecanografiado del escritor.

El trabajo de dar forma final al texto era componer las páginas del libro partiendo de páginas escritas a máquina. Redactar el texto era mecanografiarlo —literalmente, escribirlo a máquina—. Las tareas estaban perfectamente definidas y separadas. El formato de la escritura lo daba directamente la máquina. No había necesidad de mayor o mejor formato antes de la publicación. El mecanografiado tenía que estar claro para quien escribía y para quien luego necesitaba componer.

Volvamos de nuestro viaje en el tiempo. Quien escribe tiene que componer tipográficamente el texto. La máquina de escribir se ha vuelto increíblemente complicada. ¿Por qué? Porque queremos que la máquina de escribir haga también el trabajo tipográfico, que componga la página. Sin posibilidad de distinguir entre tareas. Eso tiene una aparente ventaja. A saber, conseguiremos textos con calidad de imprenta sin tener que componerlos. Aunque en la práctica no sea realmente así, la teoría promete esa posibilidad. A pesar de que no queramos, tenemos que dar formato al texto. Si no lo hacemos, será menos legible.

Hay un detalle que muestra la necesidad de formato en un procesador de textos. La auténtica labor de composición tipográfica es administrar el espacio en blanco, no las manchas negras[20]. Una letra que no sea de tipo máquina de escribir[21] es mucho menos legible sin una distribución homogénea de espacios verticales y horizontales que con ellos. No digo que sea imposible trabajar así un texto. Sin embargo, es innecesariamente más difícil. Los resultados son peores y el trabajo es más lento.

Creo que la solución pasa por volver. No atrás en el tiempo, sino volver a distinguir las tareas de escritura y de composición tipográfica. Para eso tenemos que retomar la escritura solamente. Se trata de generar un manuscrito que tendremos que entregar al tipógrafo. Imagínate que lo escribieses a máquina. Sólo tendría caracteres, como en el ejemplo del capítulo anterior. Con caracteres puedes distinguir todo para luego mandarlo a imprenta.

Escribir así se convierte en una tarea autónoma. No necesitas especificar cómo aparecerá el texto con el formato final. Sólo necesitas saber qué es cada cosa. Por si no lo hubiese hecho suficientemente, me explico. Tendrás que marcar títulos, notas al pie o énfasis. Pero no es necesario que determines cómo han de ir los títulos o las notas al pie. Eso lo puedes hacer cuando termines. O mandar al ordenador que lo haga por ti. Precisamente en eso consiste la tipografía digital.

Edición digital es el proceso por el que el ordenador se convierte en tu asistente de imprenta. Usando los programas adecuados —los veremos en la parte siguiente—, puedes hacer que el ordenador haga el trabajo de composición tipográfica. Para poder hacerlo, sólo tiene que saber qué es cada cosa. En otras palabras, sólo tiene que tener un manuscrito completo en el que no haya nada que adivinar.

En el fondo, puedes sacar muchísimo más partido a los ordenadores cuando adviertas algo básico. No son juguetes infantiles, sino máquinas a las que les puedes mandar. Por sí mismos no hacen nada, hay que ordenarles. Son asistentes extraordinariamente eficaces y velocísimos. Para ordenar tienes que hablarles, tienes que saber cómo dirigirte a ellos. Porque no

son en absoluto inteligentes, ni autónomos. Como te he dicho ya, siguen órdenes. Para que puedan seguirlas, hay que saber comunicarlas.

Los ordenadores requieren una comunicación extremadamente precisa para trabajar. Imagínate una versión muy reducida del idioma inglés, con una sintaxis también muy básica. La precisión que exige la máquina se da por la extremada limitación en la recepción del mensaje. El problema que tenemos las personas para trabajar con ordenadores no es que sean muy difíciles. En realidad, son muy precisos y extraordinariamente simples. Somos demasiado inteligentes para procesos absolutamente simples. Por eso, para manejar una máquina universal —eso es un ordenador— hay que saber cómo trabaja.

Podrás sacar todo el potencial a un ordenador cuando organices la tarea según las capacidades de la máquina. Porque aunque programases, no podrías cambiar la capacidad de la máquina. Los ordenadores son del modo en que son. Sólo podemos ordenar si sabemos cómo procesa un ordenador las cosas. De otro modo, no conseguiremos resultados. O lo que consigamos será pobre porque será muy mejorable.

Por supuesto, para trabajar con textos puedes fingir un procesador que te sitúa ante una página en blanco que contendrá el resultado final. Aunque es un modo pobre de trabajar. Porque en la simulación es en lo que un ordenador gasta más tiempo de proceso y recursos. Inútilmente, porque infrautilizas el potencial de la máquina.

Con un archivo de texto puro tienes posibilidades que no puedes soñar con un archivo de procesador de textos. En primer

lugar, todo está a la vista, no hay nada oculto. No hay problemas de compatibilidad. Da igual si usas el programa tal o el sistema operativo cual. Sólo necesitas un programa que pueda abrir archivos de texto puro. En cualquier sistema operativo tienes más de una docena. Luego, con el programa adecuado, puedes hacer cambios en ficheros muy grandes a la velocidad de la luz. Puedes también tener un historial de versiones muchísimo más completo y de acceso mucho más rápido de lo que nunca podrías haber imaginado.

Las tres posibilidades que te he mostrado en el párrafo anterior son sólo ejemplos. Intentan mostrarte por qué puede merecer la pena tener un modo más propio de trabajar con textos. Evidentemente, la decisión es tuya. Incluso para ver cómo puede ser y si realmente te compensa.

Un último consejo antes de empezar con la parte siguiente tras la propedéutica. Este modo de trabajar con textos es nuevo para ti. Tanto por la perspectiva, como por los contenidos. Creo que el aprendizaje es fácil. Aunque seguro que no es instantáneo. Necesitarás práctica y tiempo. Para hacerlo con cierta soltura, considero que deberás disponer al menos de un mes. No será a tiempo completo, aunque todo aprendizaje requiere un tiempo de digestión para llegar a saber. Mi consejo es que pruebes con tiempo. De otro modo, te puedes meter en un berenjenal de aúpa. Porque sin tiempo, no conseguirás sacar el trabajo que tengas entre manos, ni tampoco aprenderás nada relevante.

LOS PROGRAMAS

7 pandoc

pandoc es un programa que convierte texto en diferentes formatos. Su desarrollador principal es John MacFarlane. La última versión en el momento en que escribo estas páginas es 1.19. Está disponbile para *Windows*, *MacOS X* o *Linux* en `https://pandoc.org/`. Se desarrolla según el modelo de la programación informática libre. Se rige bajo las condiciones de la *GNU General Public License*[22].

La característica que más podría destacar es su velocidad. Por ejemplo, genera el documento ePub de *Ensayo sobre la escritura* en un segundo con unas algunas centésimas[23]. Con un ejemplo tomado del propio MacFarlane[24], genera un documento ePub de un manual —*Pro Git*[25]— en apenas tres segundos con tres décimas. Ambos ejemplos en un ordenador portátil que ya ha cumplido una década.

A Instalación

La instalación de pandoc es como la de otro cualquier programa en tu sistema operativo —el que uses—. Lo único que tienes que ir es a su sitio en internet y buscar dónde se encuentran las descargas. El enlace suele estar en la página de inicio. La página de instalación es `https://pandoc.org/installing.html`.

Como consejo general, a no ser que sepas lo que estás haciendo, mejor es descargar el programa para tu sistema operativo

de `https://github.com/jgm/pandoc/releases/latest`. En enlace te mostrará una página con los cambios de la última versión. Después de la lista de cambios de la versión, encontrarás las versiones para *Windows* y *MacOS X*. Descárgate la que necesites.

B La ventana de órdenes

pandoc no es un programa gráfico. No tiene una ventanita en la que haya menús y opciones. Irónicamente, te podría decir que para eso tienes el restaurante en la planta baja. No es visual. Necesita que le escribas las órdenes. Como te enseñaré en *8.A.γ. ¿Por qué un editor de textos?*, hay una manera de que puedas ordenar a pandoc que trabaje con sólo pulsar una tecla. Pero tienes que saber cómo se hace de modo no automatizado.

Para eso, tienes que abrir la ventana de órdenes. En *Windows*, se llama Símbolo del sistema y en *MacOS X* se llama Terminal. Si usases *Linux*, sabrás de lo que te hablo. Para comprobar que pandoc está bien instalado y funciona, tienes que escribir en la ventana de órdenes del sistema lo siguiente:

```
pandoc --version
```

Esa orden debe de dar como resultado un mensaje largo, que comienza con la línea que sigue. Te aconsejo que te fijes sólo en la primera línea, porque habrá más información que es lógico que no entiendas.

```
pandoc 1.19
```

El número indica la versión[26]. Siempre debes usar la última. De otro modo, te pasará que estás sufriendo fallos que pue-

den estar solucionados con bastante probabilidad. Tampoco disfrutarás de las últimas mejoras incorporadas.

Si vieses otro mensaje que te indica que no reconoce o no sabe qué es `pandoc`, me temo que no está bien instalado. En *Windows*, tienes que considerar dos cosas, que `pandoc` se instalará por defecto sólo para el usuario con que lo instales[27]. Y cuando hayas instalado `pandoc` tienes que reiniciar la sesión del usuario, para que *Windows* sepa dónde encontrar `pandoc`.

C *Primer documento de texto*

Para que pueda explicar *7.D. Manejo básico del programa*, es necesario que tengas un archivo de texto que `pandoc` pueda entender. Lo más fácil es que abras tu editor de texto favorito y escribas algo como:

```
En un lugar de la Mancha.
```

No tiene que ser un texto largo. Es sólo una prueba. Es también importante este primer archivo no tenga acentos. Es para evitar problemas tontos antes de empezar. No te preocupes, no es que `pandoc` no acepte texto acentuado. Es que en primer lugar, tienes que saber guardar tus archivos con la codificación adecuada. No es un formato de archivo, sino un formato de cómo se ordenan las letras en el archivo. Se llama conjunto de caracteres y te lo explico en *8.A.α.* Unicode.

Después de escribir el texto, guárdalo con el nombre `archivo.md`[28] en el escritorio. Como va a ser una prueba, te será más fácil acceder a él y poder borrarlo cuando ya hayas comprobado que funciona.

Puedes usar `pandoc` para convertir en muchos más formatos que el que te describo ahora. Pero como se trata de una introducción, creo que es importante que sólo nos centremos en la creación de documentos ePub. Es el estándar de publicación de libros electrónicos[29]. Con `pandoc` te centras en escribir el texto y él se encarga de generar el documento ePub cuando se lo ordenes.

Como ya te he contado, `pandoc` no es un programa visual. Su manejo es muy sencillo. En la ventana de órdenes, tienes que escribir sólo lo siguiente:

```
pandoc entrada.md -o salida.epub
```

Por supuesto, si no estás situado dentro de la ventana de órdenes en la ruta o en el directorio —lo que visualmente se llama carpeta—, antes tienes que llegar allí. Porque ahí es donde tienes `archivo.md`. En todos los casos la orden que tendrás que teclear será `cd Desktop`[30]. La orden es `cd`, que viene de *change directory* —«cambia el directorio»—. La palabra después de la orden de cambio —`Desktop`, en este caso— es a dónde quieres llegar[31].

La estructura de la orden es muy básica:

```
pandoc entrada.md -o salida.epub
```

1. Lo primero es el nombre del programa que usas.

2. Tienes que decir de qué archivo tiene que leer el texto. Pueden ser uno o varios archivos de lectura, que se separan entre sí con espacios.

3. El formato que tiene el archivo de entrada —del que se lee el texto— `pandoc` lo toma de la extensión. En el caso de `entrada.md` o `archivo.md`, la extensión es `.md`, la propia de *Markdown*.

4. Has de decir en qué archivo quieres que esté el nuevo libro. Para eso, tienes que poner `-o` antes del nombre del archivo.

5. El tipo de archivo que quieres conseguir lo especificas con la extensión del archivo de salida. En `archivo.epub` o `salida.epub` es `.epub`, que es la que identifica a los documentos ePub.

No olvides lo siguiente respecto a los nombres de archivo:

- Antes de empezar, has de saber algo básico sobre nombres de archivo. Tienen la estructura de `nombre.ext`. Como supongo que entenderás, no puede haber dos archivos con el mismo nombre completo —extensión incluida— en el mismo directorio. El último borraría al anterior.

  El nombre sin la extensión identifica el archivo. Tiene un número máximo de caracteres y existen caracteres que no puede incluir. *Windows* tiene más caracteres prohibidos en nombres de archivos y directorios que *Unix*.

  La extensión es lo que permite distinguir al sistema operativo qué tipo de archivo es. En realidad, es lo que permite que cuando pinchas dos veces con el ratón, el

sistema sepa con qué programa abrir el archivo. Obviamente, no lo sabe por el nombre, sino por la extensión.

Aunque no sea una buena idea, se pueden incluir puntos en los nombres de archivo. El sistema no va a prohibir un archivo como `tes.doc.doc`[32]. La extensión está después del último punto en el nombre completo. Con puntos extra, puedes cambiarla sin darte cuenta[33].

- Los nombres de archivo no deben incluir espacios. Si los tienen, han de ir entrecomillados.

  La siguiente orden tiene cuatro archivos —con uno repetido—, en vez de los dos archivos que debería tener:

  ```
  pandoc arch ent.md -o arch sal.epub
  ```

  Para que no haya problemas y el sistema lea dos archivos realmente distintos, tienes que escribir:

  ```
  pandoc "arch ent.md" -o "arch sal.epub"
  ```

  Por cierto, no se trata de una particularidad de `pandoc`. Ningún sistema operativo tiene manera de interpretar esos espacios como pertenecientes al nombre de archivo si no van entre comillas.

- En la medida de lo posible, evita a toda costa escribir nombres de archivos en mayúsculas.

  Excepto en *Windows*, mayúsculas y minúsculas se distinguen en nombres de archivo. En *Unix*, son distintos archivos los que tengan estos nombres[34]:

```
escritura-ordenador.md
Escritura-ordenador.md
Escritura-Ordenador.md
```

Te aconsejo que consideres muy especialmente no escribir los nombres de archivo sólo en mayúscula. Es una mala práctica, porque así son más difíciles de leer.

- El nombre del archivo debe de ser lo más descriptivo posible para que puedas conocer su contenido sin tener que abrirlo[35]. Saber nombrar es cuestión de cierta práctica. Lo perfeccionarás cuanto más lo hagas.

Los nombres de archivo no son una cuestión secundaria. Es algo fundamental, ya que puedes perder datos por hacerlo mal. Por ejemplo, si para *Windows* es lo mismo lo que para *Linux* o *MacOS X* no lo es, es fácil que puedas copiar volviendo a escribir lo que no deberías haber copiado —porque al volverlo a escribir, lo borres—. O que dejes de copiar lo que tendrías que haber copiado.

No se trata de que temas hacer algo mal. Sería tan absurdo como el niño que tuviese miedo de caerse al aprender a andar. No pienses tampoco que esto no es para ti porque es complicado. Si ya supieses, no tendrías que aprender. De la época cuando más aprendiste de tu vida, no pensaste que hablar no era lo tuyo, por muchos errores cometieses. Es la misma situación. Sólo tienes que aprender a hacerlo bien.

8 *Utilidades*

A Editor de textos

Con el método que te propongo, tienes que usar un editor de texto sin formato. Es lo que se llama texto puro. El archivo sólo contiene caracteres —letras, números y de otro tipo—. Lo único que tiene que hacer es permitirte teclear y guardar el archivo de texto en codificación UTF8 o *Unicode*[36]. No tiene nada que ver con un procesador de textos. Repito, es texto puro. Lo que significa que es texto, única y exclusivamente texto. Sin formato, que se consigue en *Markdown* —lo verás en la parte *Escribir documentos*—, con caracteres.

α Unicode

Tengo que hacerte una descripción de qué es *Unicode* y cuál es su origen. Los ordenadores tienen internamente un alfabeto, el que se usa en inglés. Un alfabeto de ordenador[37] tiene no sólo letras, sino también números y otros caracteres. Ejemplos de esos otros caracteres serían la coma, el punto o la arroba —@—. El resto de caracteres son todos los signos escritos por el ordenador que no son letras o números.

El alfabeto inglés de un ordenador cabe en 128 caracteres. Esas «letras» o caracteres hacen que *A* y *a* sean distintas. En informática es así. Cuando tienen que escribir en el resto del mundo con esos ordenadores, necesitan alfabetos de 256 caracteres. No sólo existía un alfabeto, sino varios. Las ciento

veintiocho primeras letras ya eran las del alfabeto inglés. En Europa occidental hacían falta letras como las que tienen tilde —con acentos agudo, grave y circunflejo; como *á*, *à* y *â*—, letras con diéresis —como *ä*— y letras exclusivas de determinados idiomas —*ñ*, *ð*, *ß*, *þ*, *ø*—. Esas letras especiales eran las numeradas de la ciento veintinueve a la doscientos cincuenta y seis.

Europa occidental es sólo una parte del planeta. Por eso, había varios alfabetos distintos. En Grecia, necesitaban su alfabeto, que ya habían recortado drásticamente en 1982 para hacerlo caber en la informática de entonces. En la URSS[38], Bulgaria o parte de Yugoslavia, también necesitaban el alfabeto cirílico. Por supuesto, para escribir hebreo también había que tener las letras propias.

En todos esos casos y alguno más, los caracteres que van del 128 al 256 se usaron con diferentes alfabetos. Una de las consecuencias prácticas es que sólo podías usar letras inglesas y europeas occidentales, griegas, cirílicas o hebreas. Pero no podías mezclar español —escrito completamente— y griego, o español y ruso, o alemán y hebreo. Esto es, un ejemplo tan sencillo como el siguiente era un problema:

> En griego clásico, μηδέν es «nada».

¿Por qué era un problema? Porque tenías que elegir, no se podía todo. En este caso sería, o griego, o acentos en letras latinas. En el mundo de entonces, imaginemos las obras de Marx y Engels editadas en alemán, con cometarios en ruso. O las obras de Lenin editadas en ruso con comentarios en alemán para la República Democrática de Alemania[39].

Sin casos tan lejanos o exóticos, convertir documentos —incluso con procesadores de textos como *Word*— de *Microsoft Windows* a *Apple Macintosh*[40] era problemática por los caracteres no ingleses[41]. Muchas veces, las personas sin especiales conocimientos informáticos tenían que revisar sus textos a mano.

Un modo de combinar textos[42] era teclear todo con caracteres latinos y luego usar tipografías diferentes para diferentes alfabetos. *α* era realmente *a* con tipografía *Graeca*, por ejemplo. Tenías que tener el tipo de letra, o reconocías más o menos lo que ponía[43]. Estarás de acuerdo conmigo en que no era la situación óptima.

Además, te podrás preguntar cómo escribían chinos, japoneses y coreanos. Creo que tenían sistemas propios, igual que para escribir en árabe. Realmente eso no se solucionó hasta la aparición de *Unicode*. En vez de diferentes alfabetos de 256 letras, ¿qué tal uno que permita incluir todas las letras del mundo?

Precisamente eso es *Unicode*. Aunque hay que hacer una precisión. Una cosa es el alfabeto como tal y otra lo que incluya de éste cada tipo de letra. Me explico. Como en *Unicode*, *b*, *β*, y *б* son tres letras distintas en el ordenador, no es posible la confusión. *β* y *б* no son *b* con tipo de letra para griego o cirílico, sino caracteres distintos. Cada carácter tiene una posición única en *Unicode*. Los tipos de letra digitales normalmente tienen algunos alfabetos, como latino y griego o cirílico. No siempre tienen todos los caracteres dentro de esos alfabetos definidos. Pero sobre todo, no existe una tipografía digital que contenga todos los alfabetos del mundo. Ese archivo sería inmenso. Necesitarías también muchos recursos informáticos para manejar

ese archivo en los distintos programas o con el mismo sistema operativo. En un plano puramente teórico, sería de locos, agotaría todos los recursos del sistema. Además, en la práctica es muy difícil diseñar un tipo de letra para todos los alfabetos del mundo.

β La elección adecuada

Existen muchos editores de texto puro. Los hay para todos los gustos. Tendrán más o menos posibilidades de hacer cosas. Lo importante del editor de textos es que sea el que te permita trabajar lo mejor y lo más cómodo posible. Porque si puede hacer muy pocas cosas, a la larga irás más lento. Y si puede hacer muchas pero estás incómodo, el resultado no será el mejor posible.

La elección del editor de texto con el que estés más cómodo es estrictamente personal. Depende de tu sistema operativo, de tus conocimientos de informática, del tipo de textos que vayas a escribir y de tus gustos. No por ese orden, ni tampoco es una enumeración exhaustiva. Tienes que buscar, informate, trabajar con ellos. Y encontrar un equilibrio. Que uses un editor concreto, no impide que no puedas cambiar nunca. Puedes probar y ver si te convencen otros. Ahora bien, al igual que en la vida, ir picando de flor en flor sólo manifiestaría tu desorientación.

Como anécdota que puede ilustrar, te puedo contar mi experiencia. No para que la tomes como ejemplo, sino para que veas. Durante mi vida he usado diferentes editores de texto. Cuando usaba *Windows* —en la edad oscura, hace mucho tiempo—, usé un par o tres distintos. Los nombres me temo que no recuerdo

ninguno. Cuando tengo que usar *Windows* ahora, me sirvo de *Notepad++*[44].

En *Linux*, he usado *gedit* y ahora estoy con *Geany*[45]. Con los editores profesionales de programación reconozco que no me aclaro. Son demasiado complicados para mi cabecita limitada. O requieren una paciencia y un tiempo que no les voy a dedicar —no programo—.

Los párrafos anteriores sólo relatan mi experiencia. No es extrapolable a nadie. ¿Qué es mejor o peor? Lo que sea más útil. La utilidad no es universal, porque depende de la tarea —que puede ser común— y de quién la haga. Encuentra lo que te sirva para poder sacarle el máximo partido.

¿Por qué un editor de textos?

Un editor de textos es necesario porque te puede facilitar mucho las siguientes cuestiones:

- Coloreado del texto.
- Automatización de funciones.
- Cerrado de fórmulas (o de pares de caracteres).
- Estructura del documento.

Como todo, si te empeñas, puedes usar un editor de textos básico que sólo permita escribir y grabar. Trataré de explicarte qué te perderías. O qué ganas con un editor de textos con las funciones básicas.

El coloreado de textos se refiere a que no todo el texto se muestra igual. Se usan, además de colores, negritas y cursivas para distinguir sus diferentes partes. Es sólo un recurso visual para que te sea más fácil leer lo que estás escribiendo. No es formato en el texto final. Con otro tipo de código, sin adornos:

```
<text>
  <body>
    <p>For the first time in ten years...</p>
  </body>
</text>
```

Un editor de texto que coloree muestra esto, que sin duda es mucho más legible:

```
<text>
  <body>
    <p>For the first time in ten years...</p>
  </body>
</text>
```

Ni te preocupes, ni te distraigas, porque el código de ejemplo no es de lo que tratan estas páginas. Sólo muestra la diferencia que hace que cometamos menos errores. Así sabes qué es código y qué es texto[46].

La segunda cuestión es la automatización de funciones. Lo más básico sería generar documentos en otros formatos desde el archivo que estés trabajando. Como hemos visto antes, para generar tu documento ePub, tendrías al menos que teclear:

```
pandoc -o documento.epub documento.md
```

Si lo haces del modo manual, tendrías que abrir una ventana de órdenes y teclear. Cuando tengas el documento ePub, lo podrías abrir. Si lo hicieses una única vez, sería más cómodo hacerlo a mano. Si tienes que hacerlo habitualmente, es mejor que pulsando una tecla —o una combinación de teclas— te ejecute la orden. Incluso es lógico que te abra automáticamente el documento ePub cuando `pandoc` haya terminado de generarlo.

En cualquier editor medianamente pasable —no tiene que ser profesional para programación—, es posible hacer eso. Es sencillamente que convierta el archivo *Markdown* que estés editando en documento ePub y abra este último, sólo con pulsar una tecla o combinación de teclas. Eso se haría con una orden del estilo:

```
pandoc -o "$NAME_PART".epub "$FILE_NAME" &&
mupdf "$NAME_PART".epub
```

Lo que quiere decir es que `pandoc` genere un documento ePub con el mismo nombre —pero con distinta extensión[47]— que el que estás trabajando. Y cuando termine sin errores —eso significa `&&`[48]—, pides a `mupdf` que abra el documento ePub.

Si esas dos órdenes las vinculas a una tecla o a una combinación de teclas, cuando la pulsases sólo tendrías que esperar a que apareciese el documento. Incluso podrías seguir modificándolo, si la espera fuese lo suficientemente larga como para que te diese tiempo a escribir más. Ése debe ser tu modo normal de escritura. Cuando quieras ver el resultado, que sólo tengas que pulsar y esperar a que el ordenador acabe. De otro modo, sería inaceptablemente engorrosa.

El cerrado de fórmulas no se refiere a expresiones matemáticas, sino a la agrupación de texto. Es especialmente importante con el código. Los corchetes —como veremos en *9.K. Notas al pie o al final*— pueden llegar a marcar el inicio y el final de una nota al pie. Ningún editor nos va a cerrar una nota, porque no puede saber dónde acaba. Pero sí nos puede decir dónde empieza. Esto es, nos marca dónde está el paréntesis, la llave o el corchete de inicio cuando escribimos el correspondiente carácter final. En muchos casos es extremadamente útil.

La última de las ventajas de un editor de texto medianamente bueno es que nos muestra la estructura del documento. Es algo tan sencillo como que nos da —en un lateral, muchas veces— un índice de las diferentes divisiones que tiene nuestro texto. En un editor que sólo permita escribir, es imposible verlo. O sólo podemos contemplarlo si creamos el índice en el documento que generamos. No es necesario y muchas veces habrá secciones del nivel más bajo que no quieras que aparezcan en los índices.

B Visor de documentos ePub

No es ningún secreto que `pandoc` no sólo genera documentos ePub. Genera otro tipo de documentos, pero creo la mejor introducción a `pandoc` se hace creando documentos ePub con él. Cuando generemos documentos de archivos de origen con código —realmente con cualquier el tipo de código— es bueno que comprobemos mientras escribimos cómo queda el texto en su formato final. Porque nos podemos llegar a equivocar. O podemos pensar que ciertos elementos tienen una apariencia distinta a la que tendrán.

Como se trata de documentos ePub, es necesario conseguir un visor de documentos para ese formato. Creo que las mejores opciones son *Calibre*[49] y `mupdf`[50]. Son programas totalmente distintos, ambos de código abierto. Ambos son multiplataforma. *Calibre* está pensado para ordenadores y `mupdf` funciona en *Windows*, *Linux*, *Android* y *iOS*. *Calibre* entiende las dos versiones del formato ePub —segunda y tercera—, mientras que `mupdf` sólo implementará la segunda versión del formato ePub.

`mupdf` es sólo un visor de documentos de varios formatos. Es extraordinariamente rápido. *Calibre* permite gestionar los libros electrónicos de un lector de libros electrónicos. También permite editarlos y convertirlos desde otros formatos. Mi experiencia con *Calibre* es que muestra bastante bien los documentos ePub. Pero es infinitamente más lento en esa tarea. Para las conversiones, añade mucho código inútil, que no agiliza la visualización[51] y hace engorrosa la edición del documento ePub si fuese necesaria. Como gestor de libros electrónicos creo que no es bueno si eres exigente. Convierte todos los libros —viéndole las tripas a esos libros, te das cuenta de que están mal—, incluso los que no tiene que convertir. Y es muy lento copiando libros entre el lector y el ordenador.

De todas formas, para lo que necesitas, la utilidad de visualización de libros electrónicos es muy buena. Incluso si tienes *Windows*, *Calibre* ofrece una versión portátil[52] que no necesita instalación. Puedes actualizarla sobre el mismo directorio simplemente descomprimiendo ahí la nueva versión.

Para terminar este capítulo, recuerda que es importante ver cómo queda lo que escribes. Porque puedes tener errores en tu texto —en lo que estás escribiendo—. Corregirás mejor

el formato final. Sólo así comprobarás el texto y el formato simultáneamente. También puedes equivocarte con el código *Markdown*. Igual que quien tiene boca se equivoca, quien teclea mete la pezuña de vez en cuando.

ESCRIBIR DOCUMENTOS

9 Markdown

Markdown es el código que permite escribir el texto para que pandoc pueda trabajar con él. Éste es el programa, aquél es el formato del texto. En este capítulo y el siguiente, voy a explicarte cómo se escribe en ese formato de texto.

A *Texto básico*

El texto en *Markdown* se escribe como con una máquina de escribir. No se ve el formato final, sólo el texto puro —o los caracteres, si prefieres—. Las dos reglas básicas son espacios y párrafos.

La regla de los espacios es muy sencilla. Para separar una palabra de otra, da igual que escribas un espacio o más, sólo te contará un espacio. Esto es distinto a un procesador de textos. Sin embargo, es mucho más cómodo. Aunque añadas más espacios, sólo habrá uno de diferencia. Muestro el ejemplo:

```
La temperatura es        alta.

La temperatura es alta.
```

El código anterior da como resultado:

> La temperatura es alta.
>
> La temperatura es alta.

De este modo, aunque teclees un espacio de más, no pasa nada. No se notará en el resultado final. Sólo si quisieses mostrar código, se mostrarían todos los espacios.

La regla de los párrafos dice que se forman por todas las líneas en blanco entre dos líneas con texto. Por tanto, esta regla tiene dos consecuencias. La primera es que entre dos párrafos tiene que haber al menos una línea en blanco. Al igual que con los espacios, se contará sólo una cuando sean varias líneas. La segunda consecuencia es que un párrafo puede estar formado por varias líneas contiguas.

Veamos los siguientes dos párrafos:

> Esto es un único párrafo.
>
> Y éste también es otro párrafo.

Están escritos con el siguiente código:

```
Esto
es
un
único
párrafo.


Y éste también es otro párrafo.
```

pandoc convierte los saltos de línea en espacios. Si dejamos una palabra partida en mitad de una línea, se añadirá un espacio en medio en el documento final.

B Énfasis

Markdown tiene dos formas de marcar el énfasis. La primera forma es el énfasis normal. La segunda forma es el doble énfasis. El énfasis normal se marca con un guión bajo o un asterisco. El énfasis doble usa dos guiones bajos o dos asteriscos. Como en el código:

```
_énfasis normal_ y __énfasis doble__

*énfasis normal* y **énfasis doble**
```

Daría como resultado:

énfasis normal y énfasis doble

énfasis normal y énfasis doble

Por defecto, el doble énfasis marca negrita. Sin embargo, en este libro empleo el énfasis doble para obtener la tipografía redonda. La negrita resalta demasiado un texto y es mejor no usarla. Porque acaba distrayendo la atención al concentrarla excesivamente en una palabra, expresión o pasaje.

Independientemente del uso de la negrita, la elección del marcado no sé si es muy buena. Porque un mismo signo no debe servir para dos elementos diferentes: énfasis normal y doble. La máquina lo tendrá muy claro, pero las personas podemos equivocarnos. No siempre podremos estar con la atención al máximo. Para evitar errores, te recomiendo reservar el guión bajo para el énfasis de cursiva y el asterisco —que habrá de ser doble— para negrita, si la usamos.

Un caso especial de énfasis es el que ocurre dentro de un pasaje enfatizado. Si usamos cursiva, la cursiva interior debe estar en normal. Como en el título *Der Begriff des* ius gentium *im römischen Recht*. O en un texto: «ella *no* me *lo dijo*». Para eso debemos escribir:

```
_énfasis normal _enfatizado_ de nuevo_
```

De este modo conseguiremos:

énfasis normal enfatizado *de nuevo*

Hay otro un caso especial en que el énfasis no funciona con el modo antes propuesto. Es cuando se tienen que encontrar dos guiones bajos, como en:

```
Él me dijo que _sería _ella__, no su hermana.
```

En este caso debemos escribir:

```
Él me dijo que _sería *ella*_, no su hermana.
```

Así conseguiremos un correcto énfasis desenfatizado:

Él me dijo que *sería* ella, no su hermana.

De otro modo, sin asteriscos, el resultado sería:

Él me dijo que _sería _ella__, no su hermana.

El problema está en que `pandoc` no puede cumplir órdenes que no consigue casar. El marcado funciona porque hay un principio y un final. En este caso, se trata de dos comienzos de énfasis con cursiva, que están marcados por dos guiones

bajos. Sin embargo, acaban abruptamente con una marca final de énfasis con negrita. Como no son coherentes, `pandoc` no cambia esos guiones bajos en énfasis.

C *Superíndices y subíndices*

Un modo fácil de añadir superíndices y subíndices a nuestros documentos es de la siguiente manera:

```
cm^3^, H~2~O
```

Así tendremos:

cm^3, H_2O

Por supuesto, con números, nada nos impide usar los caracteres especialmente diseñados para eso en m² o cm³ —tecleados como `m²` o `cm³`—. Pero el tipo de letra los tiene que tener definidos, lo que puede no ser el caso. Por ejemplo, la tipografía *Cousine* no tiene definido el dos subíndice —por eso no se muestra en los documentos PDF y ePub—:

```
HO, CO
```

Ni tampoco está definido para *URW Garamond*, que usa el documento PDF:

HO, CO

D *Enlaces de internet*

Para incluir direcciones de internet, tenemos dos posibilidades. El primer modo es directamente, como en el ejemplo:

```
La dirección es <https://pandoc.org>.
```

El código nos da como resultado una dirección con enlace[53]:

La dirección es https://pandoc.org.

Es muy importante advertir que las direcciones hay que ponerlas con el protocolo correspondiente. `http://` será en la mayoría de los casos. Porque de otro modo, `pandoc` no reconoce las direcciones. En ese caso, no aparecerían los enlaces en los documentos finales.

La segunda forma es poner un enlace pero con un texto distinto a la dirección de internet. Un ejemplo puede ser:

```
El programa se descarga de [su página de
internet](https://pandoc.org).
```

El resultado será el texto que hayamos escrito con un enlace a la dirección de internet:

El programa se descarga de su página de internet.

En esta segunda forma, podemos llegar a no escribir el protocolo. Aunque, como norma general, los enlaces a internet deben escribirse con su protocolo —en la gran mayoría de casos, `http://` o `https://`—. Así la máquina sabe exactamente cómo tiene que tratarlos.

E Títulos

`pandoc` permite diferentes niveles de títulos. En realidad, admite hasta seis niveles. Un título se marca con el carácter de

almohadilla —#— y un espacio al comienzo de la línea, como en:

```
# Primer nivel
## Segundo nivel
### Tercer nivel
#### Cuarto nivel
##### Quinto nivel
###### Sexto nivel
```

Los títulos deben estar escritos todos en una única línea. Esto es diferente a la regla general del párrafo. Deben estar separados por una línea en blanco del párrafo anterior. Sin embargo, si añaden una línea a la del título, no formará parte del texto. Por ejemplo, las líneas siguientes dan el resultado que muestro a continuación:

```
##### Título de prueba
temporal
```

Título de prueba

temporal

Sin embargo, si el título no tiene separación de una línea con el párrafo anterior, no se tomará como título. Formará parte del texto del párrafo anterior.

F Citas independientes

Existen momentos en que podemos querer citar un texto más extenso que forme un párrafo independiente. Al menos

uno, o pueden ser varios párrafos de cita independiente. Para eso, debemos añadir el signo de mayor que —>— y un espacio al comienzo del párrafo, como en:

```
> En un lugar de la Mancha, de cuyo nombre no
quiero acordar-me, no ha mucho tiempo que vivía
un hidalgo de los de lanza en astillero, adarga
antigua, rocín flaco y galgo corredor.
```

El resultado del párrafo anterior sería:

> En un lugar de la Mancha, de cuyo nombre no quiero acordar-me, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor.

Siempre que formen un párrafo, el signo > seguido de espacio sólo se ha de especificar en la primera línea. Cuando se trate de varios párrafos que formen una única cita, es necesario añadir el signo > en las líneas en blanco que separan los párrafos. Si no lo haces, estarás creando varias citas seguidas en vez de una única cita independiente.

Por tanto, una cita compuesta de varios párrafos debe teclearse:

```
> —Os han dicho la verdad.
>
> —¿Asististe a la acción?
>
> —Allí estuve.
```

Eso daría como resultado una única cita con varios párrafos:

—Os han dicho la verdad.

—¿Asististe a la acción?

—Allí estuve.

La diferencia sería varias citas consecutivas:

—Os han dicho la verdad.

—¿Asististe a la acción?

—Allí estuve.

El último ejemplo provendría del error de no incluir los saltos de párrafo —las líneas en blanco— en la cita, como muestra el código:

```
> —Os han dicho la verdad.
> —¿Asististe a la acción?
> —Allí estuve.
```

G *Listas*

Las listas de elementos se marcan añadiendo un asterisco —*— al comienzo de la línea. Una lista de frutas sería:

- manzanas
- peras
- plátanos
- piñas
- cocos

Vendría del código:

```
* manzanas
* peras
* plátanos
* piñas
* cocos
```

Ésa sería una lista compacta. Si quisieses una lista más suelta, debes añadir —al menos— una línea en blanco entre alguno de los componentes de la lista:

- manzanas
- peras
- plátanos
- piñas
- cocos

Lo normal será que separes en el código cada uno de los elementos de la lista suelta con una línea vacía antes del siguiente elemento. Así te será más fácil leer tu propio texto.

```
* manzanas

* peras

* plátanos

* piñas

* cocos
```

Para añadir párrafos a los elementos, tienes que introducir el segundo párrafo de un elemento con un tabulador o cuatro espacios en blanco.

```
* Manzanas.

    Pero mejor que sean de temporada.

* Plátanos.

    Por favor, no muy maduros.
```

El código anterior tendría como resultado:

- Manzanas.

  Pero mejor que sean de temporada.

- Plátanos.

  Por favor, no muy maduros.

H Listas entrelazadas

Puedes entrelazar listas del mismo modo que añades párrafos a un elemento de la lista. Sólo tienes que añadir asterisco y espacio al comienzo de la línea.

```
* Frutas

    * Fresa

    * Tomate

        Es el fruto de una planta cultivada.

* Verduras
```

```
    * Lechuga
    * Calabaza
```

El código anterior daría como resultado:

- Frutas
  - Fresa
  - Tomate

    Es el fruto de una planta cultivada.
- Verduras
  - Lechuga
  - Calabaza

Como curiosidad, el tomate es una fruta. La diferencia entre conocimiento y sabiduría es saber que el tomate es una fruta y no incluirlo en una macedonia.

I Listas numeradas

Las listas numeradas son iguales que las no numeradas, excepto porque las escribes con un número seguido de punto y espacio al principio de la línea. Un ejemplo:

```
1. Uno
1. Dos
1. Tres
```

1. Uno

2. Dos
3. Tres

Las listas se numeran automáticamente, no hemos de numerarlas nosotros. La secuencia es igual, excepto en el primer elemento. Si no es uno, la lista empezará por el número que le indiques.

2. Uno
3. Dos
4. Tres

La lista anterior es el resultado del código:

```
2. Uno
5. Dos
8. Tres
```

Como práctica general, recomiendo usar siempre el uno como numerador de todos los elementos. Ya se encargará el ordenador de numerar.

α *Otras numeraciones de listas*

En letras minúsculas:

a. primero
b. segundo
c. tercero
d. cuarto
e. quinto

Del siguiente código:

```
a. primero
a. segundo
a. tercero
a. cuarto
a. quinto
```

En números romanos en minúscula —realmente se usan en inglés, en español son en mayúscula—:

i. primero
ii. segundo
iii. tercero
iv. cuarto
v. quinto

Es resultado del código:

```
i. primero
i. segundo
i. tercero
i. cuarto
i. quinto
```

Con letras mayúsculas:

A. primero
B. segundo
C. tercero

En este caso, son necesarios dos espacios después de la letra y el punto.

```
A.  primero
```

```
A.  segundo
A.  tercero
```

Es para evitar errores en un párrafo que comience con:

```
A. Martín buscó el modo...
```

Lo mismo sucede con los auténticos números romanos:

I. primero
II. segundo
III. tercero
IV. cuarto
V. quinto

El código muestra los dos espacios después del punto y antes del texto del elemento:

```
I.  primero
I.  segundo
I.  tercero
I.  cuarto
I.  quinto
```

Por supuesto, las listas pueden entrelazarse y comenzar por valores superiores a uno:

II. primero

    a. uno

III. segundo

    d. otro

El ejemplo tiene su origen en el siguiente código:

```
II.  primero

    a. uno

I.  segundo

    d. otro
```

Por si tienes dudas, cada vez que se entrelaza una lista, cada lista subordinada es nueva e independiente para su numeración de otras listas —subordinadas o no—.

J Listas de descripciones

Existe un tipo de lista en que sus componentes se muestran según una palabra clave. Se llaman listas de descripciones o de definiciones, porque eso es precisamente lo que contienen. Se destaca una palabra o expresión y les sigue una descripción del componente. Un ejemplo sería.

> Los programas básicos para edición digital son:
>
> `pandoc` genera documentos ePub.
>
> Disponible en `https://pandoc.org`.
>
> `mupdf` muestra documentos ePub.

El código del ejemplo anterior es:

```
Los programas básicos para edición digital son:

`pandoc`
```

```
:   genera documentos ePub.

    Disponible en <https://pandoc.org>.

`mupdf`

:   muestra documentos ePub.
```

Del ejemplo anterior podemos sacar las reglas de creación de las listas de descripciones:

- Cada término de la lista —`pandoc` y `mupdf`, en el ejemplo— debe estar separado por una línea en blanco.

- La descripción se distingue porque la línea comienza con el signo de dos puntos. Tiene que haber tres espacios de separación hasta comenzar la descripción.

- No es obligatorio añadir una línea en blanco después del término que quieras describir.

- Si de un término quieres añadir varios párrafos de descripción, entonces debes añadir la línea en blanco entre el término y la descripción.

- Los párrafos de la descripción añadidos después del primero deben comenzar con una sangría de cuatro espacios.

K Notas al pie o al final

Las notas al pie se añaden con un acento circunflejo seguido de corchetes de apertura y cierre, como en:

```
Esto es una muestra^[De nota al pie.].
```

El acento circunflejo se consigue de dos formas. Pulsa dos veces sobre el acento circunflejo, o bien pulsa una vez e introduce un espacio.

Es importante que no pierdas de vista que la nota al pie existe donde haya una página. En los documentos ePub, no tienes notas al pie. Propiamente, son notas al final.

El ejemplo anterior sólo puede contener un párrafo de texto dentro de la nota. Si quisiésemos más párrafos dentro de la misma nota, el modo de escribir la nota es distinto. Has de hacerlo en dos pasos.

Primero, tienes que crear la marca de la nota. La marca tiene la forma de `[^marca]`. Esto es, empieza por un corchete de apertura y un acento circunflejo, la palabra clave y se cierra con un corchete de cierre. La nota estará donde pongas la marca.

El texto de la nota puedes escribirlo en cualquier lugar del documento. Lo mejor es justo después del párrafo que contiene la nota. Así evitarás muchos errores tontos. Empieza con la marca —en el ejemplo, `[^marca]`—, seguida de dos puntos y el texto de la nota. El primer párrafo va inmediatamente después, sin que haya línea en blanco respecto a la marca. El resto de párrafos irán sangrados con cuatro espacios o un tabulador, además de separados cada uno por una línea en blanco.

La explicación de los dos pasos te puede parecer muy ardua o demasiado abstracta. En realidad, es una tontería. Sólo tienes que cogerle el truco. Te muestro un ejemplo completo de nota al final en varios párrafos:

```
Esto es una muestra[^marca].

[^marca]: De una nota mucho más larga.

    Sólo así la nota tiene más de un párrafo.
```

La palabra clave —en el ejemplo, `marca`— es lo que permite que coincidan la nota y su texto. Debe ser distinto por cada nota. Puede contener letras, números, guiones y el signo de dos puntos.

Para notas con menos de un párrafo de texto, es mejor que uses la combinación `^[]`. En notas que ocupen más de un párrafo, la palabra clave tiene que ser igual en el cuerpo y en el texto de la nota para que la nota se genere. Para evitar errores tontos, te aconsejo que la nota de varios párrafos la coloques justo después del párrafo en el que se encuentre la marca.

L Datos del documento

Existen una serie de datos propios del documento mismo. Se llaman también «metadatos» —del inglés *metadata*—. Describen el título, subtítulo y otras características del documento. Un ejemplo sería:

```
---
title: Mi título
subtitle: Mi subtítulo
author: La autoría
date: 2016
publisher: Editorial Editora
lang: es
license: Todos los derechos reservados
cover-image: portada-documento.svg
```

```
stylesheet: hoja-estilos.css
---
```

Deben escribirse al principio del archivo y se inician con tres guiones y se terminan con otros tres guiones —o tres puntos, pero sólo finales—.

Los valores de estos campos se incluyen automáticamente en la generación de diferentes documentos. Por ejemplo, en los metadatos del documento ePub. Cada campo tiene un nombre en inglés, que paso a describir.

`title` determina el título del documento. Si bien no es obligatorio, es importante añadirlo.

`subtitle` añade un subtítulo al documento, si lo tuviese.

`author` al igual que el título, es importante añadirlo.

`date` permite añadir la fecha de generación del documento. Puede tratarse de la fecha completa o simplemente del año.

`publisher` es información opcional que especifica la editorial. Puede usarse para añadir una dirección de internet.

`lang` especifica el idioma principal del documento. Importante si se tiene la partición de palabras activada. Los valores más comunes son: español, `es`; inglés, `en`; francés, `fr`; italiano, `it`; alemán, `de`.

`license` especifica los términos de uso del documento. No es obligatorio, pero es extremadamente útil para quienes

lo usen. Aunque pueda parecer lo contrario, cuando no hay licencia alguna, se aplica la normativa vigente. Si queremos una licencia distinta, debemos especificarlo.

`cover-image` sólo afecta al documento ePub. Es la imagen de portada. Si no se especifica, `pandoc` generará una automatizada. La imagen puede ser formato JPEG, PNG o SVG.

`stylesheet` especifica la hoja de estilos del documento ePub. Si bien `pandoc` añade una hoja de estilo predeterminada, puede ser interesante poder cambiarla. Son hojas de estilo en cascada, que no trataré en estas páginas[54].

Mi consejo sobre los campos que debes añadir en cualquier documento serían: `title`, `author`, `lang` y `license`. Esto es, al menos debemos especificar título, autor, idioma y licencia del documento.

10 Elementos y atributos

En el capítulo anterior, te mostré las capacidades básicas de *Markdown*. Antes de seguir, es necesario que te explique una cuestión básica sobre el modo en que un ordenador trabaja con textos. Son los elementos y atributos.

A Elementos

Para poder escribir, necesitamos una estructura que hace posible la fijación del texto. En español —como en todas las lenguas occidentales—, además de letras, usamos palabras. También formamos líneas y párrafos al escribir. Con el ejemplo más básico: ¿cómo distinguimos las palabras en un texto? Porque ponemos espacios —además de signos de puntuación— entre ellas. Por tanto, una palabra es un elemento de nuestra escritura.

No sólo escribiendo a mano, sino también haciéndolo a ordenador, además de palabras y párrafos, existen otros elementos. Por ejemplo, los títulos de las diferentes partes del texto: partes, capítulos, secciones y subsecciones. O notas al pie de página. O listas, numeradas y sin numerar. O incluso notas al pie de página —o al final, dependiendo del formato del que se trate[55]—.

Los elementos son partes lógicas del texto. Nos permiten a personas y a máquinas reconocer qué es cada cosa en lo escrito. Especificando cada elemento para cada parte del texto, ahorramos mucho trabajo. Porque nosotros sabemos qué es cada cosa

y el ordenador sabe cómo tiene que tratarla. O podemos especificar nosotros después cómo ha de tratar cada elemento para su presentación final.

De lo que hemos visto en el capítulo anterior, son elementos: los párrafos, el énfasis, los superíndices y subíndices y las direcciones de internet. Cada uno son elementos distintos. `pandoc` sabe cómo tiene que tratar cada uno de esos elementos. Aunque podemos decirle que los trate de otra manera. Como en el caso del doble énfasis, que en vez de negrita en este documento se presenta como letra redonda.

En el siguiente fragmento de código puedes reconocer tres elementos:

```
La dirección de internet es
<https://pandoc.org>.

Este texto está _enfatizado_.
```

- El primer elemento del ejemplo anterior son los párrafos.

  Siempre hay un párrafo al principio del texto, al final y con una línea en blanco entre dos párrafos.

- El segundo párrafo es la dirección de internet. Está marcada por el signo < de apertura y el signo > de cierre.

  Además, para que `pandoc` reconozca una dirección, es necesario que tenga un protocolo de internet. En este caso es `http://`.

- El tercer elemento es el énfasis. Se marca un inicio y un fin, usando el mismo signo: el guión bajo.

Para que pandoc sepa qué es cada cosa, debes especificárselo. Eso supone que le digas qué tipo de elemento es, dónde comienza y dónde acaba.

B Atributos

Los elementos estarían muy limitados si no pudiésemos tener modos de organizarlos. Un mismo elemento puede aparecer varias veces en un texto. Por ejemplo, un párrafo, un título, una nota al pie, una dirección de internet. Cada elemento es único, aunque sus casos sean múltiples. Una nota al pie es un elemento, aunque no sea un único caso. De hecho, lo propio de las notas al pie es que están numeradas. El ordenador tiene que saber qué es una nota al pie para poder numerarla, pero no les da a todas las notas el mismo número. Igual pasa con los títulos de capítulo.

Dentro de los casos de un mismo elemento, muchas veces necesitarás distinguirlos o agruparlos. Voy a poner dos ejemplos básicos:

- Referencia a un elemento individual. Para distinguir el elemento concreto del resto, necesitamos asignarle un identificador único.

  El ejemplo más claro es una referencia cruzada, como en:

  > En la *Introducción*...

- Agrupar ciertos casos de un elemento, mediante clases.

Por ejemplo, los casos de títulos de capítulos que queremos que permanezcan ocultos. Este libro tiene títulos ocultos, como la página de derechos de autor o la del colofón[56].

- Especificar el idioma de un pasaje del texto, distinto del idioma original. El idioma es un caso especial de atributo.

  Es importante marcar el idioma original, porque las palabras no se parten igual en diferentes idiomas[57].

  Un ejemplo sencillo con *legibilidad* y *legibility*. La partición es la siguiente, cada una en su propia lengua: *le-gi-bi-li-dad* y *leg-i-bil-i-ty*. Sólo hay una coincidencia. Por eso, es esencial que cada parte del texto tenga el idioma que le corresponde.

En este punto de desarrollo, *Markdown* únicamente permite atributos en unos pocos elementos. La excepción relevante son los títulos. Aunque existe un modo de simular atributos para todos los elementos —lo veremos en *10.C. Elementos genéricos*—.

α Identificadores únicos

Los identificadores únicos son necesarios para poder hacer referencia a un determinado punto en el documento. El ordenador tiene que saber a dónde tiene que ir para poder llegar a ese destino. Eso es una referencia, que tiene la forma de enlace. Lo mismo sería si quedases con alguien: necesitas una referencia para saber dónde. No puede ser ambigua, como la puerta de entrada al parque. Porque puede haber más de un parque.

O el mismo parque puede tener más de una puerta de entrada. Un ordenador necesita una referencia exclusiva dentro del documento.

Como te he explicado en *9.E. Títulos*, un caso básico sería el elemento de título:

```
# Capítulo primero
```

Se trata de un título de primer nivel. Si nuestro primer nivel son los capítulos, será un título de capítulo. La manera de añadir un identificador único al título anterior sería:

```
# Capítulo primero {#capítulo-uno}
```

Del código anterior se deducen unas cuantas reglas:

- Los atributos van entre llaves —`{}`—, al final de la línea de título.

  No olvides que los títulos no respetan la regla de formación de párrafos. Cada título debe ocupar una única línea. Los atributos deben formar parte de esa línea única.

- El identificador único empieza siempre con el signo de almohadilla —`#`—. Se trata de una almohadilla dentro de un par de llaves.

- Para el nombre del identificador único puedes usar cualquier letra o número. El primer carácter después de `#` no debe ser un número.

- No puedes escribir espacios dentro del nombre del identificador único. El espacio separa atributos[58].

- El identificador es único en dos sentidos: no puede haber otro en todo el documento y cada elemento no acepta más que uno.

- Aunque un elemento sólo admita un identificador único, puede tener más atributos de otro tipo.

pandoc añade atributos automáticamente a todos los títulos —también el identificador único—. Por tanto, no es necesario que los añadas tú. Excepto que haya una razón especial para hacerlo, te aconsejo que no añadas identificadores únicos a los títulos. Porque evitarás problemas innecesarios, como veremos en *10.E. Enlaces dentro del documento*.

β Clases

Las clases son un tipo de atributo que permite agrupar casos de un mismo elemento[59]. En cuanto al número de casos, puede ser un único caso de un elemento, o varios. ¿Podrían ser todos los casos de un elemento? En ese caso, la clase es válida, pero es irrelevante. Esa clase es irrelevante porque es superflua. Si distingues todos los casos de un elemento, estás distinguiendo realmente el elemento. Por definición, un elemento es distinto de otros elementos —una nota al pie no es un título de capítulo—.

Supongamos que quieres que haya una serie de títulos que estén ocultos. Para ésos deberíamos especificar una clase con el nombre que queramos. Como en el ejemplo siguiente:

```
# Colofón {.oculto}
```

Las clases tienen las siguientes reglas:

- Como atributos que son —igual que los identificadores— van entre llaves, al final de la línea del título.

  Sólo puede haber un par de llaves por título conteniendo todos los atributos del título. No es específico de los títulos, todos los elementos que tengan atributos en *Markdown*, deben tenerlos dentro de un único par de llaves.

- Se especifican con un punto —`.`— y un conjunto de caracteres.

- No deben contener, igual que los identificadores, espacios en medio. Los espacios separan los diferentes atributos —sean identificadores o clases—.

- Como los identificadores, los nombres de clase pueden contener cualquier letra o número. El primer carácter —justo después del punto— no puede ser un número.

- Cualquier elemento puede tener tantas clases como queramos. No se limita a una sola, a diferencia de los identificadores únicos.

- Si bien `pandoc` genera automáticamente clases, sólo añade las necesarias para su funcionamiento. Es difícil que no tengas que añadir clases para usarlas tú. Porque las clases que genera `pandoc`, principalmente le sirven sólo al programa.

Respecto a lo anterior, si todos los títulos de capítulo —o de primer nivel— tienen que estar ocultos, no hace falta añadirles `{.oculto}` o similar. ¿Por qué? Porque tendremos que ordenar que el elemento título de primer nivel no se muestre, no sólo una clase —un grupo de casos dentro— de ese elemento. Si tienes que distinguir a todos, no distingues a ningún caso de nada. Lo realmente distinto es ya el elemento —el título de capítulo o de primer nivel, en este ejemplo—.

Un caso parecido de clase para un único caso es darle formato a un capítulo especial. Puede ser el colofón, la dedicatoria o la página de derechos de autor. Los tres ejemplos deben tener un formato distinto al resto del documento[60]. Aunque sean distintos entre sí.

C *Elementos genéricos*

Antes de poder explicarte el atributo de idioma, necesito contarte qué son los elementos genéricos. Éstos son el modo de conseguir que tengan atributos los elementos que no los tienen en *Markdown*.

El modo más fácil de entender los elementos genéricos es como contenedores. Son sólo las marcas de principio y de fin que establecen una delimitación de contenido. Esa delimitación no es un elemento concreto, como en el resto de los casos. Se trata de un elemento sin propiedades específicas, o un elemento genérico.

Puedes preguntarte para qué sirve un elemento genérico. En realidad, un elemento genérico tiene dos utilidades. La primera es la agrupación de elementos distintos para formar un

elemento nuevo. De este modo, puedes tratar como unitario algo que no lo es. El segundo uso es poder crear un elemento nuevo que sólo ha de tener una propiedad especial. Con `pandoc` y *Markdown* existe una tercera posibilidad: añadir atributos a elementos que carecen de ellos[61].

Como te he mostrado en el capítulo anterior, los elementos tienen su contenido entre el comienzo y el término de ese elemento. Para eso hay que marcar dónde comienzan y dónde terminan. Por ejemplo, con el énfasis, has de usar un guión bajo para marcar el inicio y otro para marcar el final. También en una lista, marcas el principio con el primer elemento y el final con el último elemento de esa lista. En una nota al pie de varios párrafos, el comienzo está en la expresión `[^palabra-clave]:` seguida de un espacio y el final está en el primer párrafo tras el inicio que no comienza con cuatro espacios o un tabulador. La única diferencia es que los elementos anteriores no son genéricos, tienen ya un modo predeterminado de tratar el contenido.

Todos los elementos lógicos de un texto tienen una clasificación básica. Se dividen en dos grupos, según estén dentro del párrafo o superen el párrafo. Los primeros son elementos de línea. Entre éstos, se encuentran el énfasis, los subíndices y superíndices, o los enlaces de internet. Los segundos son elementos de bloque. Párrafos, títulos, citas independientes, listas, componentes de listas o notas al pie son ejemplos de elementos de bloque.

Por tanto, los elementos genéricos son dos: uno de línea y otro de bloque. El primero es el fragmento de párrafo. El segundo es la división de bloque. Ambos carecen de propie-

dades. Sólo agrupan contenido, como te he dicho. Se vuelven útiles cuando les adscribimos un atributo, del tipo que sea. No significan nada en sí, les tienes que añadir un identificador, una clase o el idioma. De otro modo, pandoc no sabrá qué hacer con ellos. Sin atributos, tampoco las personas sabríamos para qué están los elementos genéricos.

α *División de bloque*

La división de bloque es un elemento que contiene —al menos— un elemento de párrafo. No me refiero sólo a lo que te expliqué en *9.A. Texto básico* sobre los párrafos. Es todo lo que contenga un salto de línea al final. Así son elementos de bloque, además de párrafos: títulos, listas, componentes de listas, notas al pie y citas independientes. Los elementos de bloque pueden contenerse unos a otros —aunque no todos a todos—. Por ese motivo, el elemento de bloque se denomina `<div>`. Como sabes mejor que yo, del inglés *division*.

En este punto, con los títulos, pandoc tiene una característica muy importante. Encapsula todos los fragmentos de texto con un título en un elemento de bloque automáticamente. Así no sólo podemos manejar el título, sino la porción de texto de la que es que cabeza. Gracias a esa propiedad, te librarás de tener que introducir una buena parte de divisiones en tus textos.

En este documento, hay un elemento genérico de bloque. Es la fecha que aparece en el prólogo. En realidad, se trata de un truco para añadir atributos a un párrafo[62]. El código que la genera es:

```
<div class="signature-date">
26 de agosto de 2014
</div>
```

Por tanto, para crear un elemento de párrafo tienes que observar las siguientes reglas:

- El inicio y fin del elemento genérico es `<div>` y `</div>`. Son etiquetas XML y de momento en *Markdown* no existe otro modo de hacerlo[63]

  Las etiquetas se escriben entre los signos <>. La etiqueta final incluye además una barra —`<div>` y `</div>`—.

- La clase ha de escribirse completamente, según el esquema `atributo="valor"`. En el ejemplo anterior es `class="signature-date"`.

  El único espacio permitido está entre `div` y `class`. El espacio separa el nombre del elemento y el atributo —la clase—.

- Para los dos puntos anteriores: hay diferencia entre minúsculas y mayúsculas. `<DIV>` o similar es una etiqueta inválida.

Después de estas reglas, puedes legítimamente preguntarte por qué es tan complicado algo que en el fondo sólo está alineado a la derecha. Máquinas y personas estructuramos las cosas de diferentes maneras. En el caso de textos, también es así. Decir que un párrafo está alineado a la derecha, es un modo de estructura visual. Es una manera posible de organizarlo. Porque en este caso ya sabemos que se trata de la fecha de la firma.

Lo único que pasa es que sólo nos fijamos en la alineación a la derecha. No advertimos que es la fecha de la firma porque lo damos por supuesto.

A la máquina no podemos ordenarle que ponga un párrafo a la derecha, si no sabe de qué párrafo se trata. Al menos tenemos que crear una división de bloque, para que luego le digamos al ordenador qué hacer con ella. Esa división de bloque podemos agruparla en la clase de alinear a la derecha. Puede funcionarnos, pero es fácil que nos de problemas. Porque no distinguimos bien el elemento.

El elemento estaría mal distinguido porque no decimos qué es, sino cómo tiene que aparecer. Lo último sería confundir el elemento con su resultado. Cuando especificas qué es bloque de texto, le das al ordenador un elemento lógico para que pueda trabajar con él. En realidad, lo que haces es explicitar el supuesto a la máquina. Decir qué es un determinado pasaje de texto. Así lo encontraríamos luego en el texto final. El ordenador tiene que saber qué es y cómo ha de tratarlo. No sólo cómo ha de ser el resultado. Aunque te parezca que es lo mismo.

Obviamente, no es muy práctico crear un elemento para la fecha de la firma. Porque entonces el número de elementos de un texto sería como el de las estrellas del firmamento. De hecho, no existe tal elemento en *Markdown*. Daría más problemas que los que podría solucionar.

Sin embargo, puedes añadir un elemento vacío —una división de bloque— y añadirlo a la clase de fecha de la firma. Posteriormente, mandarás al ordenador cómo tiene que tratar esa clase de elementos. Si los requisitos cambian para esa clase

de elementos, sólo le tendrás que mandar al ordenador que la presente de otro modo.

β Fragmento de párrafo

El fragmento de párrafo es un elemento de línea. Por si fuese más claro, en inglés es *span*. Es un rango, algo que tiene principio y fin dentro del mismo párrafo. No puede haber líneas en blanco de por medio, aunque en el código el comienzo y el fin ocurran en líneas distintas. Sólo tiene que estar en el mismo párrafo, según el criterio que te he explicado en *9.A. Texto básico*. Quizá el elemento de línea más común sea el énfasis.

El fragmento de párrafo es el elemento genérico del grupo de elementos de línea. Ejemplos de fragmento de párrafo son todas las posibilidades de tipos. Por ejemplo, versalitas o distintos tamaños del tipo. En ambos casos, *Markdown* no tiene elementos propios. Por eso, hay que usar un fragmento de línea. La manera de hacerlo con versalitas sería:

```
[Goethe]{.autor}, _Fausto_...
```

Nos daría el siguiente resultado, que está disponible desde la versión 1.18:

> Goethe, *Fausto*...

La regla básica es que los atributos van entre paréntesis, según la estructura `{#identificador-unico .clases}`, con las siguientes consideraciones:

- Sólo puede haber un par de paréntesis después del texto entre corchetes, que marca el fragmento.

- Los espacios separan atributos entre sí.

- Las denominaciones de los atributos pueden contener cualquier carácter, pero han de comenzar con una letra.

- El identificador tiene que ser único en el texto y en el fragmento.

  El identificador no puede repetirse en todo el texto, ni el fragmento puede tener más de de un atributo[64] —de una expresión que comience por #—.

- Un fragmento —en realidad, cualquier elemento— puede tener tantas clases como quiera.

Por cierto, los atributos de identificador y clase los creas tú, no vienen dados —los idiomas están ya dados—. Por eso, la clase podría ser `versalita` en vez de `autor`. Para el código es irrelevante. Sin embargo, te dificultas entender tu propio código más tarde, como ya te he contado[65].

D Pasajes en otros idiomas

Una característica tipográfica que aumenta la legibilidad de un texto es la partición silábica de las palabras al final de línea. De este modo se consigue que el espacio en blanco sea más o menos proporcional en todas las líneas y en todos los párrafos.

El modo de hacerlo es que lo haga automáticamente el programa. Para conseguirlo, necesitamos especificar cuál es el idioma principal del documento. Lo hemos visto en *9.L. Datos del documento*, es el campo `lang`.

Es posible que la partición automática de palabras no funcione todavía en algunos dispositivos automáticos, como los lectores electrónicos. Funciona en los modelos más nuevos, pero no en los antiguos. De todas formas, es una funcionalidad importante y es mejor añadirla en el texto. Porque no sobra, aunque el dispositivo no pueda usarla. Sin embargo, si falta, tendremos que corregir el texto y volver a generar el archivo correspondiente.

Además del idioma principal, es necesario marcar todos los pasajes que vayan en idiomas distintos. Porque cada idioma parte las palabras de una manera distinta. Las sílabas no son las mismas en todos los idiomas. Además, hay idiomas —como el inglés— que no separan por sílabas.

Para marcar los idiomas distintos del principal del texto, necesitamos unas etiquetas más completas. El ejemplo básico sería:

```
[_Weltanschauung_]{lang="de"} significa
«cosmovisión» y [ἀρετή]{lang="grc"} es
«excelencia».
```

De momento, ni `pandoc` ni *Markdown* admiten nada más sencillo[66]. En el ejemplo anterior, hay un doble etiquetado. Por un lado, está marcada la cursiva con los guiones bajos al principio y final de palabra. Además, se necesita marcar el idioma con un fragmento de párrafo con el atributo de idioma.

Espero que en un futuro no muy lejano, exista la sintaxis especial `:lang` para idioma. De este modo, los dos atributos anteriores se escribirían `{:de}` y `{:grc}`. Además, también

sería muy útil que se acepten atributos para el elemento de énfasis[67].

E Enlaces dentro del documento

Los enlaces dentro del documento no son un elemento nuevo. Se trata sencillamente de la primera parte del enlace —visto en *9.D. Enlaces de internet*—. Si quieres tener directamente la referencia de un título, sólo has de escribir el título que sea entre corchetes. El ejemplo anterior quedaría del siguiente modo:

```
En la [Introducción]...
```

Así te ahorras todos los problemas de que un enlace a una parte del documento no funcione y no entiendas por qué[68].

Conclusiones

Con este texto he querido mostrarte cómo puedes manejar un sistema sencillo de edición digital. Con una versión un poco más avanzada de ese sistema, he generado este libro y *Ensayo sobre la escritura* —será una trilogía—. Con diferentes documentos, permite tener versiones en diferentes formatos con calidad profesional.

Con la parte que te he explicado aquí, puedes obtener documentos en formato ePub. De la misma forma —con el mismo archivo de origen— puedes obtener también documentos PDF de alta calidad tipográfica. Eso te permitiría tener versiones en papel y en formato electrónico de alta calidad. Lo que es más importante: con un único archivo de origen del texto.

Los fundamentos de cómo escribir manuscritos para edición digital ya los has adquirido. Excepto detalles, ya lo sabes. Lo que queda pendiente es saber cómo aprovechar mejor el asistente tipográfico que es el ordenador. Eso supone darle órdenes para que haga más cosas con tus manuscritos. De modo que puedas imprimirlos como si fuesen libros. Tendrás edición digital y un archivo para edición de alta calidad en papel.

Sin embargo, es necesario que vayas más allá de lo básico. Estas páginas han sido una primera toma de contacto. El camino prosigue con *Aprender pandoc*. Si quieres, puedes considerarlo un manual completo de edición digital. Ahí te explico cómo conseguirás documentos completamente formateados según tus necesidades. Creo que merece la pena.

Notas

1 D. Knuth, *The TEXbook*, Addison–Wesley, p. 9.

2 `http://www.ensayo-escritura.tk`.

3 Para que no haya incomprensiones, aclaro dos puntos. De un portátil se desgastan por uso la batería y el disco duro.

En mi ordenador, la batería la he cambiado el año pasado —no cargaba desde hace años—. El disco duro lo cambié hace unos cuatro años, cuando empezó a dar fallos.

Los discos duros pueden fallar y perder todos los datos que contienen. Como aviso, sale más barato cambiar el disco que perder todos los datos.

4 *Fedora 24 Desktop Edition*, la última versión disponible cuando escribo estas páginas (`https://getfedora.org/`).

5 Por supuesto, cambiaré de ordenador. Aunque será por mis necesidades, no por exigencias de la versión de *Windows*.

6 *Fedora 24 LXDE Spin* (`https://spins.fedoraproject.org/lxde/`).

7 La garantía en España de aparatos nuevos es de dos años, según el artículo 123.1 del Real Decreto Legislativo 1/2007 (`https://www.boe.es/buscar/act.php?id=BOE-A-2007-20555#a123`).

Realmente, la ley establece seis meses de garantía, porque el defecto —«la falta de conformidad»— debe existir «cuando la cosa se entregó». Se suponen que ya existían en los seis meses posteriores a la entrega del producto. Con una batería, es casi imposible demostrar que el fallo está en origen después de los seis meses.

8 En algunos programas de generación de documentos PDF, se omite la incrustación de tipografías por error. Es cierto que el tamaño del

documento es menor, pero se verá mal en otros ordenadores. Habitualmente la persona que genera el documento PDF no advierte este detalle por una mala configuración predeterminada de *Adobe Acrobat*.

9 Disponible en `http://www.adobe.com/devnet/pdf/pdf_reference_archive.html`.

10 En realidad, lee más formatos, como ePub, MOBI, CHM, XPS, DjVu, CBZ, y CBR. Está disponible en `http://www.sumatrapdfreader.org/`.

11 Lo único en que puede dar problemas es con la impresión en tamaños de papel no estándar. Es una parte del programa que todavía está pendiente de desarrollo.

12 Mucha gente piensa que *Windows* o *Word* —ambos de la casa *Microsoft*— son un estándar. Es debido a lo mucho que se ha copiado. Sin la «piratería», la empresa de Redmond no sería hoy lo que es.

13 Acrónimo de *hypertext transfer protocol*.

14 Para que no tengas dudas, no lo desarrolló por primera vez *Apple* para sus ordenadores *Macintosh*, sino *Xerox*. En 1973, lanzó *Xerox Alto*, modelo experimental que se comercializaría en 1981 como *Xerox Star*. *Macintosh* llegó al mercado en 1984.

15 Son órdenes que funcionan en *Unix* —el antecesor común a *Linux* y *MacOS X*—. En *Windows*, me parece que `cp` no funciona así.

16 En *Unix*, el bucle sería:

```
for i in disco/*;
do cp -i "$i" /mnt/mp3player/disco;
done
```

En *Windows* es similar, pero no igual.

17 En español son capacidades, no competencias.

18 «Fuente» es un anglicismo, no es correcto en español. El inglés *font* viene de *funditum*, no de *fons* —del que proviene nuestra «fuente»—. *Font* refiere a que se trata de tipo metálico obtenido por fundición.

En realidad, *font* es el tipo que tiene una forma —normal, cursiva, negrita, versalitas— y un tamaño concreto en puntos. Tipografía es *typeface* o *font family*.

Con un par de ejemplos para aclarar:

- *Palatino* normal, negrita, cursiva y negrita cursiva son cuatro tipos —*fonts*— distintos, pero una única y misma tipografía —*typeface* o *font family*—.

- *Palatino*, *Helvetica* y *Times New Roman* son tres tipografías distintas —*typefaces* o *font families*—.

No es informática, es terminología de la imprenta clásica.

19 Como es de justicia citar: «τὸ γὰρ αὐτὸ ἅμα ὑπάρχειν τε καὶ μὴ ὑπάρχειν ἀδύνατον τῷ αὐτῷ καὶ κατὰ τὸ αὐτό»; Aristóteles, *Metafísica*, IV 3, 1005b 19–20.

20 "You are designing not the black marks on the page, but the space in between"; E. Spiekermann, *One Thing Only*, en `http://spiekermann.com/en/one-thing-only/`.

21 Con serifas, como *Times New Roman* y *Garamond*, o redonda, como *Helvetica* y *Univers*.

22 Disponible en `https://www.gnu.org/copyleft/gpl.html`.

23 En realidad, la generación de `ensayo-escritura.epub` dura dos segundos porque es necesario activar un añadido —técnicamente es un filtro—, para no incluir los comentarios en el documento final. He de reconocerte que es el doble de tiempo en sentido aritmético.

24 Descrito en `https://pandoc.org/epub.html`.

25 Disponible en `https://progit.org/`.

26 El mensaje completo puedes verlo en `https://asciinema.org/a/8xqd771x8qkp8js9yquhjldnp`. No olvides que es *Linux*, en otro sistema operativo será un poco distinto.

27 Para instalar en todo el sistema a todos los usuarios, tendrías que ejecutar la orden `msiexec /i pandoc-1.xx.msi ALLUSERS=1`.

Tienes que sustituir `xx` por el número de versión que se corresponda con el archivo que descargues.

28 El uso del guión de separación debajo de una palabra con tipo de letra mecanografiada es el modo de marcar que la palabra hay que escribirla sin ningún guíon, ni normal, ni bajo.

Añado la partición de palabras también para el código —las partes que están mecanografiadas—, porque sería muy difícil escribir estas páginas de otro modo.

La partición silábica del código con guión bajo sólo aparece en el documento PDF —o las páginas impresas con ese documento—.

En el documento ePub no se parten las palabras, porque la mayoría de programas que leen ese formato no son capaces de hacer partición automática. Aunque pueda haber partición silábica normal de palabras en documentos ePub, no es posible cambiar el carácter y el modo en que se hace esa partición.

29 *Amazon* usa para sus libros un formato propio. En realidad, el formato de la empresa estadounidense es muy similar al formato ePub.

30 Por si no funcionase, la orden completa en *Windows* es:

```
cd %USERPROFILE%\Desktop\
```

En *Unix*, la orden completa sería:

```
cd $HOME/Desktop
```

Depende de dónde esté situada la ventana de órdenes, podrás llegar con `cd Desktop`. Si no fuese posible, usa la orden anterior correspondiente a tu sistema operativo.

31 Puede ser que no sea ese el directorio, por diferentes motivos. En ese caso, tendrás que buscarlo tú. No te preocupes, porque es imposible que pase nada en el sistema cambiado de directorio. Aprenderás a moverte dentro de la ventana de órdenes, que es importante saber hacerlo.

32 Además de que es bueno no usar abreviaturas para nombres de archivo. Es más fácil leerlas rápido y mal. Si las quieres usar, no uses puntos, sino guiones —incluso bajos—.

33 La opción predeterminada en muchos sistemas operativos es ocultar las extensiones conocidas de archivo. Una extensión es conocida para el sistema, cuando tiene un programa asignado para abrir los archivos que la tengan. Visualmente, las extensiones conocidas se distinguen porque los archivos tienen un icono propio.

Por ejemplo, `.doc`, `.odt`, `.pdf` o `.html` marcan archivos de *Microsoft Word*, *LibreOffice Writer*, *Adobe Reader* y el navegador de internet del sistema, respectivamente. Tendrán el icono propio en sus archivos.

Las extensiones desconocidas siempre se muestran —como `.md`—. Hasta que le digas al sistema que la tiene que abrir *siempre* con un editor determinado. Después siempre abrirá los archivos `.md` con el editor elegido, porque ha asociado la extensión.

Con la opción activa de ocultar de extensiones conocidas, es imposible cambiarlas fuera de la ventana de órdenes. En el entorno de ventanas es imposible, porque no se muestran nunca. Aunque hay que aprender, porque puedes dejar inservible un archivo si le pones la extensión errónea. No podrás abrir ese archivo con la extensión cambiada, porque no tendrá programa asignado —o el programa que lo abra, no podrá hacer nada con el archivo—. Con muy mala suerte, puedes perderlo, porque escribas datos que estén en otro formato. Aunque creo que es muy difícil porque es fácil que el mismo programa te diga que no entienden el archivo, o que está dañado. En todo caso, *nunca grabes un archivo si ves que se abre con otro programa o que se muestra muy distinto a la última vez que lo cambiaste*.

34 En *Windows* se trataría de uno y el mismo mismo nombre.

35 He usado `arch ent.md` y `arch sal.epub` en la orden para que esté completa en una línea. Serían nombres de archivos bastante malos.

36 Técnicamente UTF8 y *Unicode* no son la misma cosa. Pero para lo que necesitas, es como si fuesen sinónimos.

37 No es una expresión técnica, es un modo de explicarlo. Las denominaciones técnicas son juego de caracteres o código de página.

38 Rusia no existía entonces, era la parte más grande de la Unión de Repúblicas Socialistas Soviéticas. Ucrania también usa el alfabeto cirílico. Pero hasta 1991 fue otra de las repúblicas socialistas soviéticas.

39 No es tan descabellado como podría parecer. En el bloque de países comunistas, marxismo–leninismo era una asignatura no sólo universitaria. Por ejemplo, Angela Merkel —actual cancillera federal alemana— tuvo que demostrar suficientes conocimientos de esa materia para su doctorado en Química con el trabajo escrito *Was ist sozialistische Lebensweise? —¿Qué es un modo de vida socialista?—*. No era la excepción, el marxismo–leninismo era la ortodoxia de esos países.

40 Nada de *MacOS X*, que llegó mucho más tarde.

41 Por si no hubiese sido claro, fundamentalmente eran eñes y letras con acento gráfico de cualquier tipo. Porque podías tener palabras o textos en italiano, francés, o incluso alemán.

42 Lo he vivido con el griego clásico y no deja de ser una situación muy precaria. Visto desde la situación actual, no dudo que es una chapuza. Aunque haya gente que por ignorancia siga usando esta técnica.

43 Ví un ejemplo de una conocida con su tesis doctoral. Escrita en español con bastantes fragmentos en griego clásico, se la imprimieron sin el tipo de letra para griego. Empezó la lectura de la tesis —su examen, para que me entiendas— con una mala impresión que comentó todo el tribunal.

44 No uso *Geany* en Windows porque tuve problemas para la escritura de caracteres no latinos en *Unicode*.

45 No te incluyo las direcciones de internet. Si te interesan, búscalos. Aunque mejor, léete el artículo de comparación de editores de texto puro de la *Wikipedia* en inglés: `https://en.wikipedia.org/wiki/Comparison_of_text_editors`.

46 En el documento PDF el código no está coloreado, sino en tonos de gris. Sencillamente se debe a que es un gasto absurdo imprimirlo en color. Imagínate que está en azul marino.

47 Si tuviese el mismo nombre y extensión sería el mismo archivo. Es decir, la escritura del documento ePub borraría el archivo *Markdown* de origen.

48 Tanto en *Unix* como en *Windows*, `&&` en la ventana de órdenes indica que ejecute lo posterior sólo si lo anterior se ha ejecutado correctamente. `&` sería el modo de hacer que se ejecute la orden siguiente, independientemente del resultado de la anterior. Ambos conectores encadenan órdenes.

49 `https://calibre-ebook.com/`.

50 `https://mupdf.com`.

51 Los lectores de tinta electrónica son realmente ordenadores con muy pocos recursos. Cualquier gasto inútil se paga en lentitud en el manejo del libro.

52 `https://calibre-ebook.com/download_portable`

53 La dirección de internet está formateada con tipografía mecanografiada por mi decisión. La dirección de internet tiene como característica el enlace al documento electrónico. Esto es, si lo pinchas, te lo abre en un navegador.

54 Espero poder tratarlas en otro libro más completo: *Aprender pandoc*, todavía en proyecto.

55 Los documentos ePub no forman una página física. Debido a que carecen de pie de página, no pueden tener notas al pie. En esto se distinguen del formato PDF, que refleja una página física.

56 Puede también haber referencia automática: *[Colofón]*. Es necesario aclarar dos cosas.

El primer asunto es que la referencia está en los enlaces a la página. En los dos formatos que tienes de este texto: PDF y ePub. La referencia es totalmente automática en ambos formatos. Pero en el documento PDF, la referencia está más procesada. Porque al título del capítulo se le añade numeración donde el titulo correspondiente la tenga. En el documento ePub, no hay numeración que añadir al título.

La segunda cuestión es que los corchetes pertenecen al título oculto del colofón. Por eso se muestran en la referencia automatizada que genera `pandoc`. De hecho, el código de enlace interno es `[[Colofón]]`. En este caso, los corchetes internos son parte del título.

57 Los programas de lectura de documentos ePub incluyen progresivamente partición de palabras al final de línea. Por eso es fundamental especificar cuál es el idioma de cada parte del documento.

58 Con un ejemplo: `{#único .oculto}`. El espacio separa el identificador `único` y la clase `oculto` de ese elemento.

59 Nada impide que uses una misma clase para elementos distintos. Mejor evítalo, porque puedes confundir elementos por error.

60 En ambos casos, supera lo que quiero contarte en este libro. En el caso de títulos de capítulo ocultos, como en el ejemplo citado, el código del archivo de hoja de estilos del documento ePub sería:

```
.oculto > h1 {
    display: none;
    }
```

Un diferente formato para el colofón sería en ejemplo completo —con márgenes de página y tamaño del texto—:

```
.colofón, #colofón {
    padding-top: 40%;
    margin-left: 10%;
    width: 80%;
    line-height: 115%;
    text-align: justify;
    font-size: 95%;
    text-indent: 0%;
    }

.colofón p, #colofón p {
    text-indent: 0%;
    padding-top: 0%;
    text-align: center;
    }
```

Para que el código anterior funcione, es necesario que el título sea:

```
# Colofón {.colofón}
```

Hay dos posibilidades. O el título es *Colofón*, entonces `pandoc` genera automáticamente el identificador `{#colofón}`. O has de teclear la clase `{.colofón}`. Como ya he dicho, `pandoc` no genera este tipo de clases —ya genera el identificador único—.

El contenido abordado en esta nota se trata en *Aprender `pandoc`*.

61 Si bien no deja de ser hacer de necesidad, virtud.

62 El elemento de párrafo no tiene atributos en *Markdown*. Al menos, de momento.

63 Esta incidencia se lleva discutiendo desde hace más de cinco años: `https://github.com/jgm/pandoc/issues/168`. Esperemos que en breve tenga una solución.

64 En realidad, es lógico que si el identificador va a ser único en el texto, ningún elemento pueda tener más de un identificador. No lo necesita.

65 Si por lo que sea, luego tuvieses que dejar que el autor fuese normal, no en versalitas, podrías llegar a tener que escribir todos esos elementos genéricos de nuevo. Si lo haces claro desde el principio, lo tendrás claro siempre.

66 En el futuro, puede que lo hagan. Depende de que la propuesta de `https://github.com/jgm/pandoc/issues/895` se acepte.

67 Con la propuesta doble, el ejemplo sería:

```
_Weltanschauung_{:de} significa «cosmovisión» y
[ἀρετή]{:grc} es «excelencia».
```

Por si dudas, las expresiones extranjeras se ponen en cursiva si usan el mismo alfabeto. Con otro alfabeto, se ve claro que es una expresión extranjera. Se distinguen en los mismos casos que la lengua principal del documento —énfasis dentro del pasaje, títulos o similares—.

68 Lo único que tienes que hacer es escribir dentro de los corchetes el título exactamente igual que como está en el lugar en que aparece.

Este libro se ha generado con
pandoc (https://pandoc.org/).

La versión PDF está compuesta
tipográficamente con ConTEXt
(http://contextgarden.net/).

Las tipografías usadas para el documento
PDF son: *URW Garamond*, *Theano Didot*,
URW Classico y *Cousine*. *Tempestiva* y
Cousine se emplean en el documento ePub.